High Availability for High-End Scientific Computing

A Dissertation
Submitted In Partial Fulfilment Of
The Requirements For The Degree Of

MASTER OF SCIENCE

In

NETWORK CENTERED COMPUTING,
HIGH PERFORMANCE COMPUTING AND COMMUNICATION

in the

FACULTY OF SCIENCE

THE UNIVERSITY OF READING

by

Kai Uhlemann

14. March 2006

Supervisors:
Prof. V. Alexandrov, University of Reading
Stephen L. Scott, Ph.D., Oak Ridge National Laboratory

Acknowledgment

I would like to thank all of those who have given their time, assistance and patience to make this dissection possible.

For making this thesis, and my research at the Oak Ridge National Laboratory possible, I would like to thank my advisor, Stephen L. Scott, and also, Christian Engelmann. I appreciate their invitation to do my Master's thesis work at such a renowned institution, and for the financial support that allowed me to achieve it.

I am particularly greatful to both Stephen and Christian for their moral support, and for sharing my excitement in ongoing research.

Thanks to Ronald Baumann, who provided both private support and amazing debugging abilities, and to Cindy Sonewald, who struggled with administration and bureaucracy to help me survive my stay in the US.

Abstract

With the growing interest in and popularity of high performance cluster computing and, more importantly, the fast growing size of compute clusters, research in the area of high availability is essential to meet the needs to sustain the current growth.

This thesis introduces a new approach to high availability which focuses on the head node of a cluster system. This projects focus is on providing high availability to the job-scheduler service, which is the most vital part of the traditional Beowulf-style cluster architecture.

This research seeks to add high availability to the job-scheduler service and resource management system, typically running on the head node, leading to a significant increase of availability for cluster computing.

Also, this software project takes advantage of the virtual synchrony paradigm to achieve Active/Active replication, the highest form of high availability.

A proof-of-concept implementation shows how high availability can be designed in software and what results can be expected of such a system. The results may be reused for future or existing projects to further improve and extent the high availability of compute clusters.

Contents

1 Introduction

1.1 Project overview

1.1.1 The idea of High-end scientific computing

High-end scientific computing (HEC) has become a powerful and important tool for scientists world-wide to investigate complex research problems like nanotechnology, nuclear fusion or the human genome project. [ES05]

HEC outlines the element that combines state-of-the-art computational powers of high performance computing (HPC), the representation of huge amounts of data and eventually the scientists working in collaboration groups in a global scale.

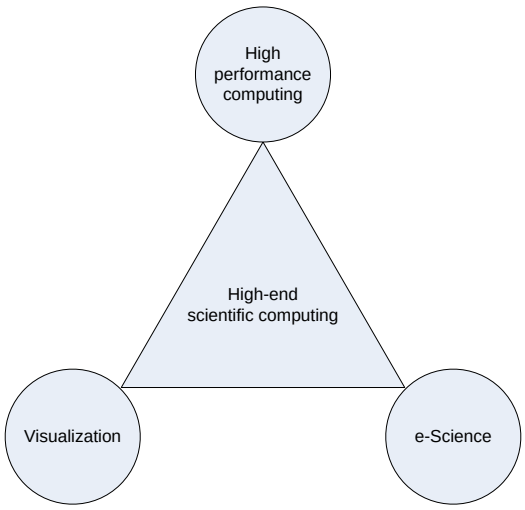

Figure 1.1.: High-end scientific computing[Rob05]

Figure 1.1 illustrates high performance computing, visualisation and e-Science connect to produce high-end scientific computing. HEC enables scientists to gain new insights by accessing, analysing and evaluating the massive amounts of data generated. Via e-Science it is possible to effectively join computation power, infrastructure and scientists. High-end scientific computing provides high performance computation in a scientific environment. [Rob05]

Generally speaking, HPC includes any application, which is impossible to run on a simple workstation due to the amount and complexity of code and algorithms. Therefore, high performance computation environments are either vector machines, massively-parallel-processing machines (MPP) or compute clusters. [Wik05b]

A cluster is a group of loosely-coupled computers that work together closely. Individual computers may be connected through fast local area networks. Typically, clusters are used to improve both speed and/or reliability in comparison to a single computer. The size of clusters can range from a couple of compute nodes up to several thousand. [Wik05a]

1.1.2 Traditional cluster setup deficiencies

Traditional Beowulf-style Linux clusters are the common setup for high-end scientific computing at research institutes all around the world. The advantages are their relative low price and easy deployment.

Nevertheless, one of the core design problems with the conventional Beowulf-style Linux clusters is that they have only a single head node manage and control the connected compute nodes. The very nature of this architecture relies on the head node as single point of failure and control.

When this particular system component fails, a single point of failure causes a system wide failure. If the head node fails, none of the healthy compute nodes connected to it, can be reached. It is also possible to loose job already running or queued. Thus, improved speed and reliability may be entirely lost due to limited availability.

Another problem is that the huge growth in performance and scale of new HEC sys-

tems emerging on the markets every year pose a considerable challenge to the software and applications running on large-scale, high-performance machines. [ES05]

Compared to their predecessors, the reliability and availability of recently deployed HEC systems has decreased with the significantly increasing number of processors. This leads to unacceptable low availability; some systems can only provide a mean time between failures (MTBF) of as low as 40 to 50 hours. [ES05]

For comparison: One of the world's first computers, the Electronic Numerical Integrator and Computer (ENIAC) from the University of Pennsylvania, had a mean time between failures greater than 12 hours. [The04b] A modern hard disk has about 1.2 million hours MTBF. [Wes05]

Quite often, the runtime for high-performance computing applications exceeds the mean time between failure rate of the hardware for the entire cluster system. [MNS+03]

Therefore, in order to take advantage of the growing computational capabilities of modern hard- and software efforts to improve the availability of HEC systems are of vital importance . Research in high availability focused on cluster head nodes is especially important to increase the overall availability of cluster computing environments. The clustered system must provide ways to eliminate single points of failure and control. [MNS+03]

1.1.3 High availability

In the past, high availability (HA) computing was only thought important and cost-effective for industry applications such as telecommunication carriers. It has now become extra important to the very foundation of high-end scientific computing and its high performance applications. [MNS+03]

Generally speaking, high availability (HA) is a system that can mask certain defined faults from the user. [Res96]

The transparency is achieved by redundancy strategy. When one of the components fails, a redundant component takes over. By using high availability approaches the

mean time to recover a system can be decreased, the possible loss of system and application state can be significantly avoided, and single points of failure and control can be prevented. [ES05]

The high availability models are distinguished from one another by the degree of transparency. The levels of high availability depend on those models and their replication strategy. Table 1.1 gives a brief overview of the different levels of high availability. [ES05]

HA models	**Active/Standby**	**Active/Active**
redundant components	single	multiple
redundancy	Standby	Active
examples	Active/Cold-Standby Active/Warm-Standby Active/Hot-Standby	Asymmetric Active/Active Symmetric Active/Active

Table 1.1.: Models and levels of high availability

1.1.4 Project problem description

This dissertation develops a proof-of-concept prototype job-scheduler service, which provides Active/Active high availability for high-end scientific computing.

One way to achieve this kind of high availability is to use component state replication with an active system component group with advanced commit protocols, such as distributed controls, or the virtual synchrony model. This paradigm can provide a job scheduler or similar essential services, with a service-level symmetric Active/Active high availability. [ES05]

The Active/Active approach for high availability is distinguished from other forms of HA such as Active/Standby solutions, by using multiple active redundant system components as replication strategy. [ES05]

I introduce a solution to tightly-couple active replicated, redundant service components in groups to make them logically indistinguishable and transparent to users.

Thus a user sees the required component behaviour as a single, running service rather than as a group of job schedulers. [Res96]

The states of all the participating processes in the group are actively and completely shared and fully replicated. If a group member fails, none of the users experience an interruption of the provided job-scheduler service. All remaining active system components of the service group remain uninterrupted and continue to run from their current global application state, so that running jobs will not be interrupted. [ES05]

The remaining members of the group still provide the job-scheduler service. The virtual synchrony model provides a complete masking of faults, and transparency to the user as well as to services as the job scheduler. [Res96]

Altogether, the virtual synchrony model means that multiple processes provide the same service by acting as one visible system component. Active replication can be achieved by using this synchronisation paradigm to share process states between all the participating processes.

Symmetric active replication refers to the symmetry between the replicated group processes. All group members process the same group communication messages as input to achieve synchronisation of the internal process state. This process state is actively and completely shared among the group components. There are two ways replication can be done: [Res96]

- internal replication
- external replication

Internal replication allows each process to independently accept an external state change using group communication messages. Therefore, since that all processes of the service group receive the incoming state requests in the same order, the requirements to achieve virtual synchrony are met.

In order to assure virtual synchrony using external replication, the external state change requests are ordered globally and then forwarded to all individual service processes using a separate process group. Since all processes of the service group

receive the incoming state requests in the same order, virtual synchrony is achieved.

Internal replication is the most effective way of providing virtual synchrony, since it requires the least synchronisation. However, the disadvantage is that existing code of a job-scheduler service module would need extensive code modification in order to implement necessary changes to achieve high availability, especially when moving from a single to multiple head-node architecture.

In contrast, external replication synchronises the service group based on sequential incoming requests which makes the synchronisation less efficient. On the other hand, external replication needs little or no code modifications to existing code of service modules, and is therefore easier to implement. That way, it is easy to reuse existing software and merely extend it with the virtual synchrony to achieve high availability.

Both internal and external replications must provide a solution for the single-instance-execution problem. In contrast to a single-headed job scheduler, which receives a job, schedules it, puts it into execution on a compute node, a multi-headed scheduler must deal with the fact that, in order to prevent multiple execution of the same job only one of the head nodes may initiate and finish the execution of a job so that valuable computation time on the cluster is not wasted.

This problem evolves from a partial asymmetry between the job scheduler group processes. Some of the tasks cannot be done by all the processes. Only one node executes the next job in the job queue on the computation nodes.

This project develops a distributed job-scheduler service which removes the single point of failure and control from single-headed Beowulf-style Linux clusters by introducing a new multi-headed cluster architecture in order to achieve Active/Active high availability.

1.2 Previous work

Previous and related research in this area includes cluster management systems solutions for Active/Hot-Standby high availability, as well as research about algorithms providing virtual synchrony including projects to develop frameworks for reliable

group communication. By using these kinds of group communication systems, Active/Active high availability can be realised.

1.2.1 Related research in high availability

Modern systems for cluster management provide resources for easy installation and management of a cluster system. In fact, the availability and serviceability could be improved because of less downtime necessary for system management. Open Source Cluster Application Resources (OSCAR) [Ope05] is a well-known example of such a system, and was developed by the Open Cluster Group, a collaboration of major research institutions and technology companies led by the Oak Ridge National Laboratory (ORNL), the National Center for Supercomputing Applications (NCSA), IBM, Indiana University, Intel and Louisiana Tech University. [LML+05]

There are several ongoing research projects related to high-availability solutions for cluster systems, such as Lifekeeper [Ste05], Kimberlite [Mis05] and Linux failsafe [Sil05]. [LML+05]

However, these solutions for high availability do not focus on the Beowulf-style cluster architecture and do not provide improved availability and serviceability support for scientific computing, such as a highly available job scheduler. [LML+05]

This gap is filled by the HA-OSCAR project [HA-05], a production-quality HA cluster solution from the Computer Science department of Louisiana Tech University. Its goals include non-stop services for Linux high-performance computing environments. [LML+05]

The project's main goals focus on Reliability, Availability and Serviceability (RAS) for the cluster HPC environment. That way, HA-OSCAR provides both high availability and high performance for Beowulf-based cluster solutions. [LML+05]

HA OSCAR also provides critical failure prediction and analysis capabilities. The initial model of choice to approach higher availability is an Active/Hot-Standby solution. So, the project only provides a policy based failover operation. [LML+05]

However, plans for HA OSCAR are to extend the initial architecture to support an

Active/Active high availability model after the release of the Active/Hot-Standby distribution. The Active/Active solution will provide much better resource utilisation, because both head nodes will be simultaneously active. [MNS+03]

1.2.2 Group communication for virtual synchrony

A certain kind of group communication is necessary in order to implement an Active/Active high availability model solution that realises the state replication between virtual synchronous processes.

Several past and ongoing research projects focus on such group communication system, fault-tolerant process groups, and highly available distributed virtual processes. [ESI02]

A group communication system in distributed computing relies on the common Internet standard protocols (TCP/IP, UDP/IP). The grade of message consistency may vary from complete virtual synchrony to accepted inconsistencies. Other models use the distributed agreement paradigm. There, all members of a process group must agree to a consistent state, after a time of inconsistency. [ESI02]

Group communication solutions like Isis, Ensemble [The99a] or transis [The99b] already exist, which provide a configurable framework for a highly-available application. [ESI02]

The Isis Toolkit was one of the first group communication systems, capable of providing at least causal and total order of message delivery. It was developed by Cornell University during the late 1980s. The Isis group communication system was one of the first systems to introduce the basic techniques for structuring distributed systems and was originally designed to simplify the development of fault-tolerant distributed systems, already based on the concepts of process group and virtual synchrony. [Bar98]

Transis [The99b], a group communication toolkit developed at Hebrew University of Jerusalem, also provides reliable multicast communication and services for the replication of information. Following the transis system model, a group member may fail

but later recovers. It also supports network partitioning and subsequent remerges. The advantage of transis as a group communication toolkit is its modularity and extensibility. [Bar98]

Thus, an existing group communication layer can be used and configured to fit application requirements and hardware features. A disadvantage is the complex combination of protocols and the use of deep protocol stacks, which could significantly reduce the performance. [ESI02]

1.3 Key problems and specification

Research in the area of high availability for cluster systems particularly focused on the head node, is still an ongoing effort. The Active/Hot-Standby solution from the HA-OSCAR [HA-05] project is an example of how to handle a failing head node.

A solution as proposed in this dissertation concerning symmetric highly-available Active/Active job-scheduler service and resource management has not yet been realised. The proposed solution may be of vital importance to the further development of solutions based on high-availability models using the active replication approach. Since the outlined service system should provide basic job-scheduler capabilities, a new service system has either to be designed and implemented from scratch, or an existing open-source scheduler must be reused and modified in order to provide the necessary scheduling and resource facilities. The decision for one of these options is notably influenced of how the replication will be realised.

If replication is done internally, the effort to facilitate an existing open-source job scheduler with high availability will be extensive and therefore, favours a design from scratch. On the other hand, by realising the replication externally, existing software would only have to be wrapped up in order to augment the service with Active/Active high availability.

The most important element for enabling high availability is to guarantee the shared global state between the service processes via the group communication facilities. In order to get global state knowledge, every possible input event to the process group

must be handled properly. To resolve the key problems of seamless recovery and dynamic service group reconfiguration, the proper reactions to those inputs should be examined and developed.

Another key problem is the single-instance-execution problem. Since the system design implies several separate group members which share the exact same process state, each of the participating job-scheduler processes would attempt to independently execute the current job in the queue. Proper facilities should be developed to prevent the job duplication.

The main goal of the proposed solution is the design, implementation and test of prototype software, which uses an existing group communication framework to meet all necessary criteria for virtual synchrony to enable active replication.

This framework must provide built-in awareness of membership state changes to enable recovery and fault detection for the group communication. The system is thus able to provide the needed global knowledge among group members concerning their membership and communication. The chosen framework must have facilities that ensure consistency of the messages among the group members to enable the virtual synchrony paradigm.

In order to ensure global knowledge between all participating processes, the process states must be actively replicated between the distributed processes. This must be achieved by the communication facilities of the chosen group communication framework, by passing messages among the group members. The framework ensures that every message sent is received in the same total order by all members of the distributed scheduler service group.

The novel approach to enable Active/Active high availability via active replication is of vital importance to this thesis. I intend to improve the existing conventional Beowulf Linux cluster system architecture and software system, in order to remove their single points of failure and control. It is then possible to bring forward high-end scientific computation needs. With the proposed solution speed, reliability and availability can be better combined to provide an exceptional way for modern cluster computing. Therefore, the service scheduler to develop provides dynamic reconfiguration and recovery from failure, completely transparent to the user. In the end,

the intended service system may be used to augment existing cluster systems, like OSCAR [Ope05], with high-availability scheduler abilities completely invisible to the users of the system.

General ideas on possible solutions will be investigated and discussed later, but only one specific scenario will be implemented to show how the concepts of Active/Active high availability work in practice.

The ultimate solution will include prototype software to proof the concepts set forth in this dissertation. Using the results and findings, it should be possible to develop a production-type scheduler service or improve existing high-availability solutions, by shifting to a higher model of high availability.

Compared to the Active/Standby model, the model of Active/Active high availability has the advantage that multiple redundant components of the service processes will be possible. Therefore, restrictions such as only one standby server in the HA-OSCAR project [HA-05] will not apply. In fact, if multiple numbers of redundant active components are possible, it is feasible to combine as many components as needed to satisfy the availability needs of the desired service application.

For the job-scheduler service, basic scheduler features must be provided by the proposed software system, including a solution for consequential side effects, such as the single-instance-execution problem.

The key features of common job-scheduler services, such as the popular Portable Batch System (PBS) [Alt03] provides, must provided by the proposed system as well. Thus the most important features for the user are:

- submit a job via command line interface using execution scripts
- get status information of currently running and queued jobs
- remove a job from the queue

But just providing basic scheduler capabilities is not sufficient. The service must also be highly available.

This Master thesis project is, therefore, about the development and implementation

of a proof-of-concept prototype job-scheduler service, which provides Active/Active high availability with the following overall system design key features:

- seamless failover in case of the failure of a head node
- dynamic reconfiguration and recovery of the service group processes are masked from the user
- complete transparency to the user in case of a fault
- high transparency of HA capabilities from the user trough command line tool replacements
- continuous availability of the scheduler service
- any amount of redundant head-node services (1+n) to scale availability

The design and implementation of the proposed job-scheduler service must provide the following high-availability capabilities, which are distinct from common solutions:

- queue commands done on or to one of the head nodes (e.g. via command line)
- no loss of scheduled job
- no loss of running job
- no restart of running jobs
- uninterrupted job-scheduler-service availability

In terms of Active/Active high availability these key features mean that the work in progress of the job-scheduler service continues by using the remaining redundant components in case of a fault. The masking of all faults must be complete and transparent to the user. Any client of the system using the job-scheduler-service facilities remains uninterrupted. [Res96]

Since job schedulers and cluster resource management systems already exist, it is better to reuse these implementations, especially since these components are not key

parts of the new system design in terms of high availability. Therefore, I investigated ways to augment an existing software component with the desired availability features.

Consequently the following objectives are achieved by this Master thesis project with descending importance:

- integration of an appropriate group communication framework providing virtual synchrony
- implementation of facilities for internal or external replication to achieve a global application state
- proof-of-concept development of a basic job-scheduler service, that takes advantage of group communication facilities to eventually provide Active/Active high availability
- solution for the single-instance-execution problem

Eventually, this Master thesis project aims to provide a system design and proof-of-concept implementation of a job-scheduler-service system with basic scheduling capabilities, providing Active/Active high availability by using an existing group communication system as basis.

1.4 Software system requirements and milestones

Following the key problems and project description, the software to develop must meet certain requirements in order to ensure that the outlined goals will be achieved. Different milestones help to evaluate every step during the development process toward the final proof-of-concept implementation.

Three milestones define the status of project progress:

- Milestone A - **minimal** criteria and requirements are met
- Milestone B - **optimal** criteria and requirements are met

- Milestone C - all requirements are met, including **extra** capabilities

The following requirement table formed the base criteria, by which to judge the success of later implementation. The system tests in particular will prove, whether all the requirements are met by the dissertation project.

Required Capability	**Milestone**
user command for job submission by script	**A**
user command for jobs deletion by job identification	**A**
job-scheduler service stays available (submission of jobs), as long as one head node is up	**A**
job-scheduler service availability without failover time	**A**
job-scheduler service processes actively replicate incoming user queue commands	**A**
user command for job queue information status query	**B**
user commands and job-scheduler service are configurable by config file	**B**
solution for single-instance-execution problem	**B**
new job scheduler processes can dynamically join	**B**
job-scheduler service may crash, without interrupting currently running job	**B**
if one component of local job-scheduler service fails, all components shut down	**B**
HA capabilities replcae user commands transparently	**B**
user command for job submission by standard input	**C**
job-scheduler service is interchangeable (if external replication)	**C**

Table 1.2.: Requirements and milestones overview

2 Preliminary system design

The design and realisation of the proposed project forms the subject of this chapter. The developed software system will implement a JOb Scheduler for High availability Using Active replication (JOSHUA).

A precise statement of system concepts will distinguish this project from related work. As mentioned before, the eventual implementation of the job scheduler must be highly available trough active replication. The following figure (2.1) shows the life cycle of a job in case of a head-node failure, using different job-scheduler services.

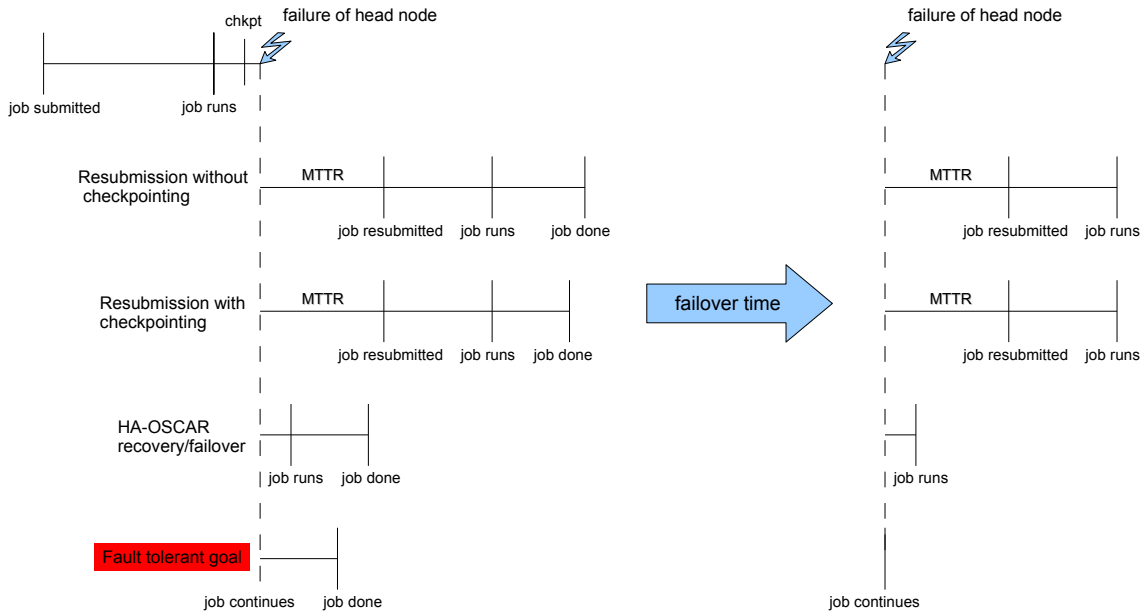

Figure 2.1.: Impact of head-node failure to job lifespan

When a computational job is done using a resource management and job scheduler

system, the job moves through several stages until it is finished:

1. submission
2. queue (waiting for execution)
3. execution
4. completion

Figure 2.1 shows the scenario of a head-node failure on four different job scheduling systems, while the job is in running state. The first two solutions use resubmission in case of a head-node failure. In that situation, all information about submitted, scheduled and running jobs will be lost.

After mean time to repair (MTTR), all lost jobs must be resubmitted in order to be executed. Neither of the first two systems provide any higher form of high availability concerning a head-node crash, although the second solution uses checkpointing to restart a rescheduled job after repair. That way only the part after the checkpoint has to be rerun to finish the job.

This kind of behaviour has a huge impact on the failover time needed for the system until the failed job gets into the running stage again. Both of the first two solutions need a total failover time of $t_f = t_{MTTR} + t_{resub}$. Especially dependent on MTTR, which can be a matter of days, the time of unavailability is unacceptably high.

The third solution from the HA-OSCAR project [HA-05] is a tremendous improvement in comparison to the first two, since the mean time to repair is not applicable in this case because of the cluster architecture design.

The HA-OSCAR project [HA-05] provides a Active/Hot-Standby solution, where a standby head node takes over in case of a failure of the primary head node. This greatly improves the time for failover to matter of seconds, since this solution needs only the time to get the redundant server online. A manual failover with HA-OSCAR only takes $t_f = 3 - 5s$. [LSL+03a]

However, I wished to avoid a failover time, and therefore the concept of failover in general. As figure 2.1 shows, a symmetric Active/Active high-availability job-

scheduler service will be a first solution for a fault-tolerant job scheduler. In this solution no failover is necessary at all, since, by definition, the service is provided by the remaining service processes within the group in virtual synchrony. The proposed system design will eventually lead to this kind of fault-tolerant job-scheduler service availability.

2.1 System design approach

The solution for the design of the proposed system uses concepts directly developed from the environment of high-end scientific computing and the applications and conventions for high-performance computing already in use. Thus an Active/Active solution is attractive because it migrates easily from former solutions.

The development of the proposed job-scheduler service takes existing and common solutions under consideration to come up with an equivalent replacement. As stated in the project specification, the HA job-scheduling capabilities should be highly transparent to the user and therefore act like already established mainstream job scheduler and resource manager. In order to accomplish that, common solutions, systems and related work will be investigated to finally lead to a final project system design.

2.1.1 Traditional Beowulf cluster system architecture

The traditional Beowulf cluster system architecture design consists of two major parts. First, the head node, which contains all necessary facilities and services to manage cluster jobs, and second, the compute nodes, which are used as execution hosts for distributed jobs such as application using PVM [GBD+94] or MPI [SOHL+96].

To take advantage of this system architecture design, a resource and job management system is used to control computational power. A commonly used environment, is a PBS (Portable Batch System) based system such as OpenPBS [Alt03], PBS Pro [Alt05] or PBS TORQUE [Clu05a].

A PBS-based scheduler and resource management system running on Beowulf system architecture usually consists of 4 major parts:

- PBS job server (pbs_server)
- PBS job executor (pbs_mom)
- Job scheduler
- User command line tools for queue manipulation
- Administrator command line tools

A basic example of a common Beowulf cluster setup is shown in figure 2.2.

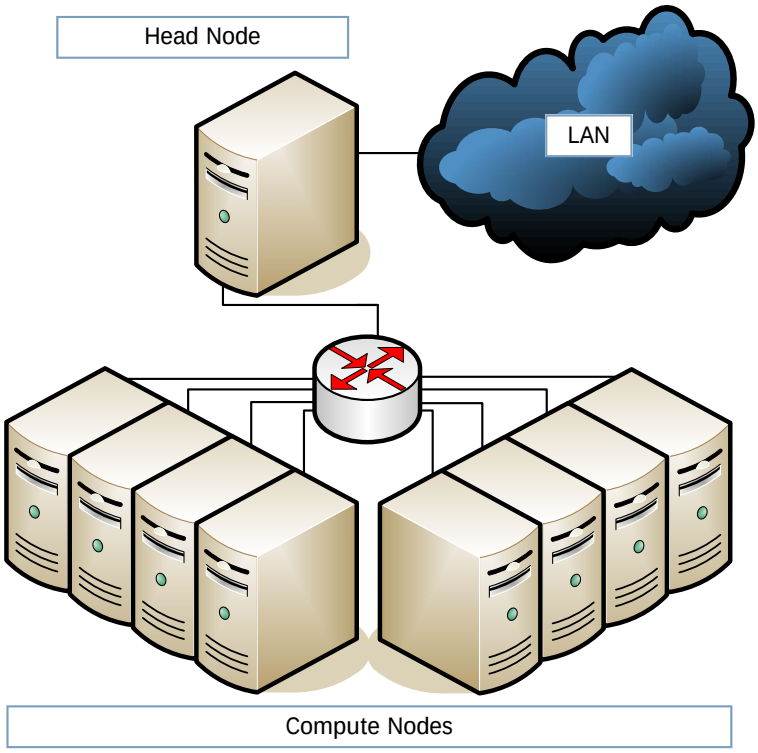

Figure 2.2.: Traditional Beowolf Architecture [LML+05]

Both PBS server and job scheduler are running on the head node as part of the resource management service system. Therefore, the head node is the central point of control of a PBS. Every user and administrator command, as well as any part of the PBS communication, goes through the server application, usually connected by an IP network. [Ver00]

PBS is responsible for providing basic batch job-scheduler services such as receiving/creating a job, modifying a job, protecting the job against system crashes (optional e.g. checkpointing), and eventually, running the job and placing it into execution on the execution nodes. [Ver00]

The central PBS server on the head node does resource management for submitted jobs and uses a scheduler to schedule jobs on the dedicated compute nodes. From there the PBS job executors put the current job(s) into execution.

In fact, the job executor actually places the job into execution. It is also informally called PBS mom as it is the mother of all executing jobs. PBS mom places a job into execution when it receives a copy of the job from the PBS server. For the execution itself, PBS mom creates a new session as identical to the user login session as possible. If needed, the job executor also provides the capabilities to return the standard output and standard error output of a job to the user. It is even possible to run a job in an interactive mode. [Ver00]

Depending on the scheduling policies and resources needed for a job, the job scheduler and PSB server fit the job into an optimal execution slot. When a job is executed, it is eventually sent from the PBS server to the PBS mom to run on the compute nodes.

Though this would be the most usual setup for a portable batch system, it is sometimes necessary and to run the PBS executors alongside the PBS server on the head node, especially when the computational power of the head node is needed. This, the head node also becomes a compute node.

By the design of a portable batch system, it is the job scheduler which is responsible for controling the scheduling and access policies of a site, to decide which job is running, and where and it runs on the compute nodes. PBS also allows the use of external schedulers, like the common Maui Cluster Scheduler™. The basic scheduler *pbs_sched* is also part of the portable batch system, but only provides a first-in, first-out (FIFO) scheduling policy, and therefore does not utilises the cluster's resources well. [Clu05b]

Usually there are four different kinds of access policies for the compute nodes managed by the Maui Cluster Scheduler™[Clu06]:

Policy	Description
SHARED	Tasks from any combination of jobs may utilise available resources
SINGLEUSER	Tasks from any jobs owned by the same user may utilise available resources
SINGLEJOB	Tasks from a single job may utilise available resources
SINGLETASK	A single task from a single job may run on the node

Table 2.1.: Access policy overview for Maui

In order to organise the use of the cluster environment and put site policies into practice, a job scheduler, as a central point must communicate with the various PBS moms to gain knowledge about the state of the system resources. In cooperation with the PBS server, the scheduler learns about the availability of jobs waiting for execution. [Ver00]

Usually the manipulation of the queue is done by command line tools for job control from the client side. The administration tools are mostly used directly on the head node.

Most importantly command tools provide the capabilities to submit, monitor, modify, and delete jobs. Generally speaking, there are three different kinds of commands: [Cor00]

- user commands, e.g. *qsub*,*qstat*
- operator commands, e.g. *qenable*, *qrun*
- manager or administrator commands, e.g. *qmgr*, *pbsnodes*

Figure 2.3 illustrates how the different components of a common implementation of portable batch system interact with each other.

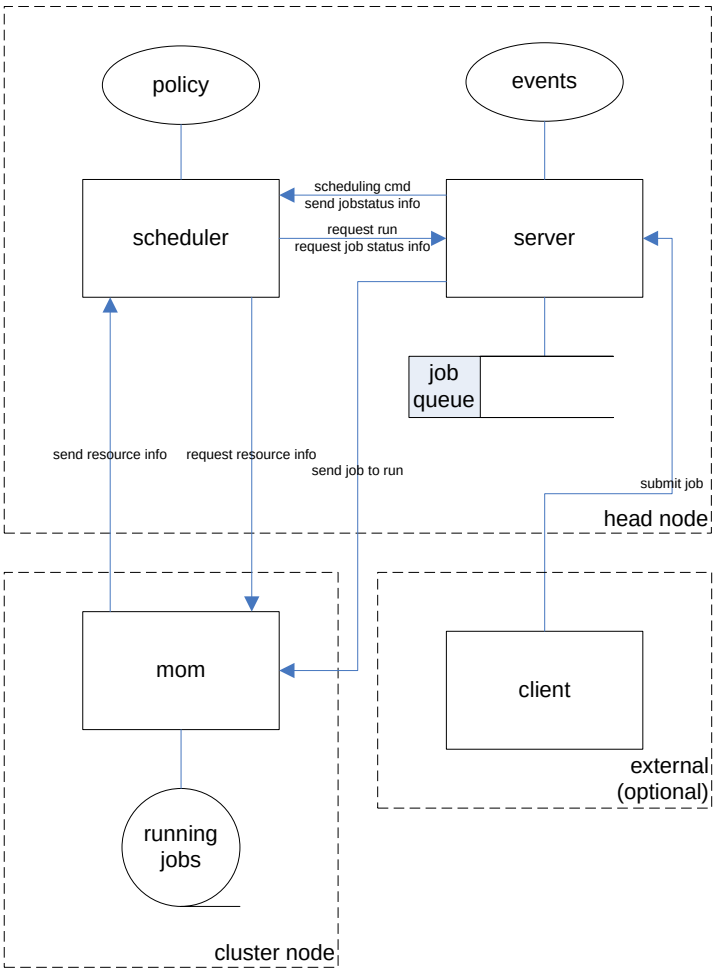

Figure 2.3.: PBS components overview [Cor00]

Computational jobs managed by PBS often have the following properties and capabilities important for the scheduler to put the policies into practice: [Cor00]

- indicator for batch or interactive job
- list of required resources
- definition of priority
- definition of time of execution
- information on whether to send a mail to the user when execution starts, ends or aborts

A list of required resources for the execution can be specified by each job. Depending on the execution environment and hardware platform used, the number and types of resource indicators may vary. It is therefore possible to set job-specific policy rules, such as: [Cor00]

- cput: max CPU time used by all processes in the job
- mem: max amount of physical memory used by the job
- walltime: wall clock time running
- host: name of the host on which job should be run

For quite some time, the portable batch system on a conventional Beowulf cluster architecture has been a common solution for computational defiances and tasks in high performance computing. Unfortunately today's requirements and demands regarding high service availability of job scheduling and resource management systems are not sufficient for high-end scientific computing.

2.1.2 HA-OSCAR cluster system architecture

To improve the conventional Beowulf cluster architecture, the HA-OSCAR project introduced an advanced cluster systems architecture.

The HA-OSCAR project aims particularly for enterprise and mission-critical systems in need for high-availability features for the Open Source Cluster Application Resources (OSCAR) [Ope05], which is a widely adopted open source Linux PC cluster system. [LSL+03b]

OSCAR is a toolkit for easy deployment of the best-known methods of building, programming and using a high performance cluster. The resource packages consist of a fully-integrated and easy-to-install software bundle, designed for HPC cluster computing. Everything needed to install, build, maintain, and use such a Linux cluster is already included in the OSCAR suite. [Ope05]

HA-OSCAR, which includes all software packages of OSCAR, introduces various enhancements and additional features to OSCAR. Those are mostly focused to the areas

of availability, scalability, and security. The key features of the latest release are head-node redundancy and self-recovery for hardware, service, and application outages. [The04a]

As figure 2.4 illustrates, the cluster system architecture of HA-OSCAR introduces head-node redundancy by adding an additional standby head node.

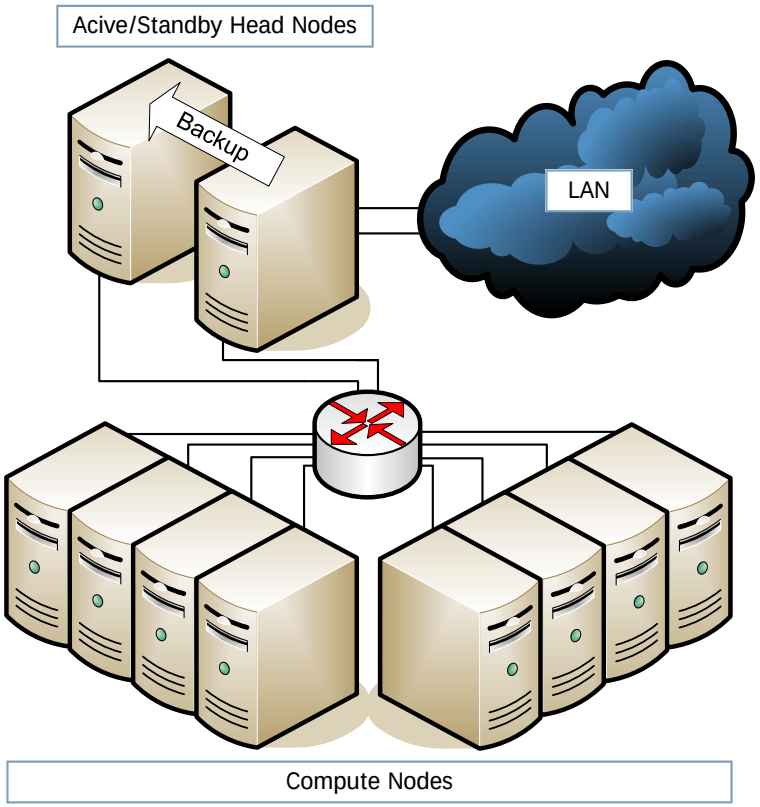

Figure 2.4.: HA-OSCAR cluster architecture using standby backup

The major system components of HA-OSCAR are [The04a]:

- **Primary server**
  - each server supports up to three NICs
  - one is connected to the Internet
  - the other two are connected to a private LAN (primary Ethernet LAN and a standby LAN)

- **Standby server**
  - monitors primary server
  - takes over the primary server when a failure of the primary server is detected
- **Compute nodes**
  - dedicated nodes for computational purposes

for further network redundancy improvement, as well as external reliable storage, HA-OSCAR will even support failover for dual private network interfaces. [The04a]

In order to remove the single point of failure and control from the typical Beowulf and OSCAR cluster system setup, HA-OSCAR provides a second standby for failover scenarios. [LSL+03b]

A reliable interconnection between the two head nodes, realised with a Heartbeat [Lin05] mechanism over both an IP and serial network link, works as health detector for the system status of the primary server. Thereby, the secondary server is able to observe the working server, and can take over in case of a failure. [LSL+03b]

In that case, the HA-OSCAR system can automatically transfer the service controls to the failover server, by allowing only a minimal interruption of service availability. [LSL+03b]

After a failover, all applications continue to run on the standby server while the main server is being repaired, until the primary server is ready to retake control. The secondary standby server is always idle while the primary server is in service. [LSL+03b]

HA-OSCAR does failover by cloning the IP Address of the primary head node. The secondary server takes the alias IP address of the failing head node in order to take over control, so that the compute nodes can continue to operate seamlessly. [LSL+03b]

Inspite of the improvements that the HA-OSCAR project is able to provide in terms of high availability of head nodes and their services, this solution still interrupts ser-

vices, though briefly.

2.1.3 Symmetric Active/Active HA for job-scheduler services

In order to avoid an interruption of availability of services on the head nodes, the approaches to high availability should be improved even further than by the HA-OSCAR project.

Preventing an interruption is the model of Active/Active high availability, which introduces multiple active, replicated, redundant components. [ESLH06]

The concept of Active/Active high availability is built on the use of the virtual synchrony paradigm. Further improvement of high availability is accomplished by using multiple redundant head nodes providing job-scheduler services running in virtual synchrony.

By using this model, head-node failures do not need a failover to a standby server and there is no disruption or loss of service at all. As long as one member of the service group in virtual synchrony is alive, service is provided. [ESLH06]

The model of symmetric Active/Active high availability illustrated in figure 2.5 allows more than one redundant service to be active, i.e. to accept state changes. If a service process fails, the system is able to continue operation using a replica. [ESLH06]

This solution does not waste system resources by relying on an idle standby. Furthermore, there is no interruption of service and no loss of state, since active services run in virtual synchrony without the need to failover. Single points of failure and control are eliminated as well. [ESLH06]

Active service state replication is realised by using a group communication system to totally order all state change messages, and maintain reliable delivery to all redundant active services processes. This group communication system must ensure total message order and reliable message delivery, as well as service group membership management. [ESLH06]

In addition, a consistent output is produced by all active services. For example, mes-

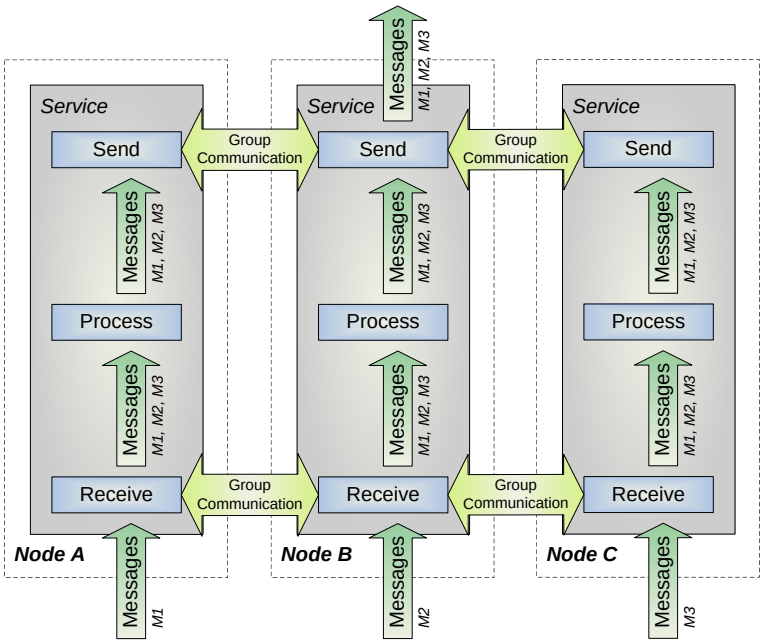

Figure 2.5.: Active/Active high availability architecture for services

sages are sent to other parts of the system or return messages of service state changes. Those messages are routed through a group communication system using it for a distributed mutual exclusion to ensure that the output is delivered only once. [ESLH06]

The enhancement of cluster system architecture with symmetric Active/Active high availability greatly improves the overall availability of such a system. As long as one active job service process is alive, process states are never lost, state changes can be performed within the group, and an output is produced according to state changes. [ESLH06]

2.1.4 Group communication system

A proper group communication must be used in order to achieve the proposed virtual synchrony needed for symmetric Active/Active high availability.

The virtual synchrony, first established in the early work on the Isis group communication system is no longer available. The group communication facilities needed should also be open source to make necessary adjustments possible.

Free projects now available are transis [The99b] and openais [The05].

The openais project is a production quality implementation of the SA Forum [Ser05] Application Interface Specification. Openais implements current research on virtual synchrony to provide a 100% correct operation in the case of failure with excellent performance characteristics. The Availability Management Framework (AMF) API of openais provides application failover, cluster membership (CLM), checkpointing (CKPT), event (EVT), messaging (MSG), and distributed locks (DLOCK). [The05]

Unfortunately, openais is still in a rather early development phase, too far from being used by this dissertation. Fortunately transis can provide all necessary group communication facilities needed for the implementation of the high available job-scheduler service system.

The transis group communication framework provides:

- group communication daemon
- library with group communication interfaces
- group membership management
- support for message event-based programming

Distributed locks or even distributed mutual exclusion solutions are not included and must be implemented.

2.1.5 Multi-head node system architecture

The concept of symmetric Active/Active high availability and the potential use of multiple actively replicated redundant components introduces a further design improvement for cluster architecture.

Instead of only one head node or one head node and one standby server, a multi-head node cluster system architecture consists of any number of equal head nodes bound together in virtual synchrony, as figure 2.6 illustrates.

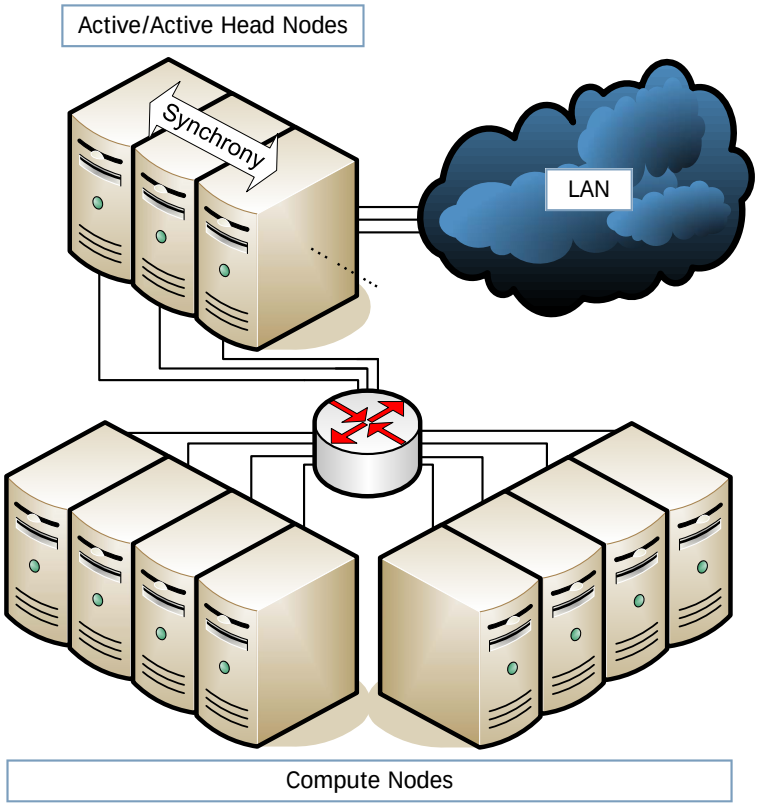

Figure 2.6.: Active/Active cluster architecture using multiple head nodes

In difference to the HA-OSCAR project solution, a multi-head node system architecture does not take redundant network into consideration, as well as network partitioning and remerging.

The preliminary system design derives from changing the architecture of a conventional Beowulf cluster to a multi-headed symmetric Active/Active cluster architecture to allow augmentation with a higher model of high availability.

Since the focus of development is on the high-availability capabilities, the replication of the redundant components will be done externally. That not only gives more time on the focused issues of that work, but also allows reuse of as much software and software libraries as possible, which are not directly involved in solving the problem of implementing the proposed form of high availability.

Reuse of a common PBS-like service for external replication meets the requirement

to produce a highly transparent HA replacement for broadly used job scheduler and resource management systems. The system design of JOSHUA therefore takes advantage of an existing job-scheduler service system, but is not restricted to the one described in this dissertation. The underlying resource management system can be easyly replaced by a similar solution.

In fact, the most important advantage in reusing an existing job scheduler application in connection with external replication is that modification of the existing code is unnecessary, because the service processes will be wrapped up by the Active/Active high availability service architecture. [ESLH06]

However, Portable Batch Systems are only meant to work in a single-headed cluster architecture environment. To enable a multi-head node setup as proposed for this symmetric Active/Active high availability for job-scheduler service, a version of PBS must be used, which is aware of the special multi-headed environment.

While this dissertation was being written, PBS TORQUE, a next generation PBS system by Cluster Resources Inc.[Clu05a], derived from Open PBS, introduced the possibility of using more than one head node. This applicable feature has been enabled since version 2.0p1 to make work in the area of high availability even possible.

As figure 2.7 shows, for PBS TORQUE, input for service processes must be intercepted, totally ordered, and reliably delivered to the service group daemon using a group communication system that mimics the service interface. This is done by the use of separate event handler routines. [ESLH06]

For example, the command line tools for queue manipulation of the job and resource management service system PBS TORQUE are replaced with an interceptor command that behaves exactly like the original, but will forward all command input to an interceptor group of service processes.

Once all messages are totally ordered and reliably delivered, each interceptor group member calls the original command locally to perform the requested operation. Then, the service group output is rerouted through the interceptor group for a single delivery. [ESLH06]

The concept of external replication implies a very coarse-grain synchronisation. An

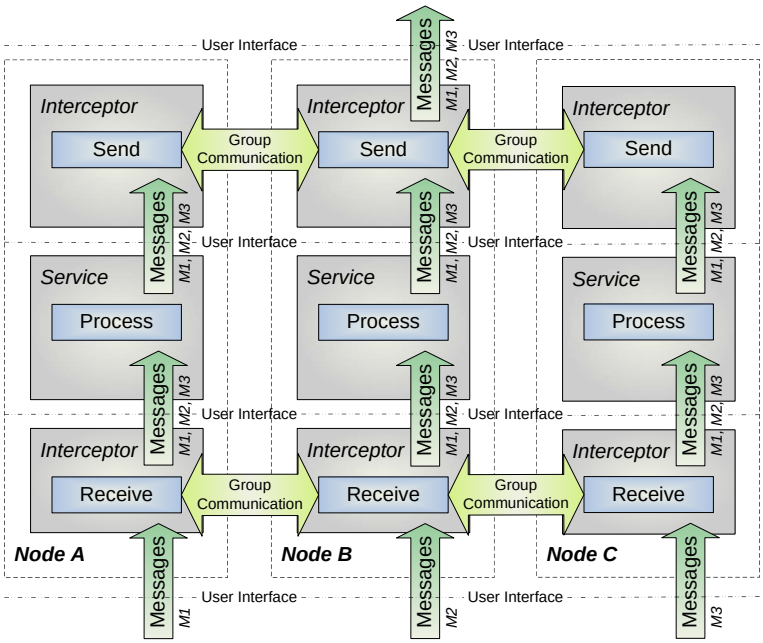

Figure 2.7.: Active/Active high availability using external replication

Interleaving state change is impossible, since the interceptor group will force atomicity for all service interface operations. [ESLH06]

The PBS TORQUE version for the service processes makes it possible for the first time, running each PBS server on multiple head nodes to communicate with multiple PBS moms on the compute nodes. In all previous version of PBS TORQUE [Clu05a] or similar PBS systems, the PBS executor daemon processes allowed only one PBS server to access the provided resources of the cluster node. In a multi-head node environment, only one of the PBS servers would have access to the compute nodes.

Multi-server support alone does not enable higher availability, but is necessary to make the proposed symmetric Active/Active high availability for job-scheduler services even possible. All connected PBS servers can now run jobs on the PBS moms.

Since this Master thesis project achieves high availability through active replication, all PBS servers must maintain the same job queues with identical job descriptions in their local queues.

The PBS TORQUE system does not provide any means to provide replication of job

queues, nor does it prevent multiple executions of identical jobs (single-instance-execution problem).

These missing features and virtual synchrony among the job service processes of the PBS TORQUE must be provided by the JOSHUA software system as illustrated in figure 2.7.

A basic system components overview of the JOSHUA system design is illustrated in figure 2.8. JOSHUA therefore consists of three major parts:

- JOSHUA server daemon (*joshua*)
- JOSHUA user commands (*jcmd*)
- JOSHUA cluster mutex (*jmutex*)

The central focus of JOSHUA is its server daemon. This component is responsible for taking all client requests and redirecting them to the local running PBS server. The server daemon acts as a gateway between the client command tools and the PBS Server and therefore transparently hides the newly introduced high-availability design from the user.

The second part of JOSHUA is its user commands bundle (jcmd). These are the replacements for the PBS user tools. To achieve the transparency to the user, the new command tools work exactly like the original PBS TORQUE user commands.

The third and last component of JOSHUA is its cluster mutex. Jmutex is a distributed mutual exclusion facility that solves the single-instance-execution problem of jobs on the cluster nodes.

The JOSHUA server daemon mimics the service interface of PBS TORQUE and locally performs all user command requests intercepted by the replacement tools by using an event-driven interface. Since all server daemons run in virtual synchrony, all user commands are totally ordered and reliably delivered to the PBS server daemon.

The output interceptor for the PBS TORQUE service system is the cluster mutex. It ensures, that only one logically output, in this case the execution command for the

current job, is intercepted, and delivery occurs just once, to avoid multiple execution of jobs.

The concurrently running PBS TORQUE servers on each head node are not aware of each other. There is no communication between the job scheduler and resource management service processes outside the communication provided by the interceptors.

To simplify the side effects of redundant PBS TORQUE servers for the cluster mutex implementation, the used Maui Cluster Scheduler^{TM}must be set up in a dedicated mode, where each job gets exclusive access to all the available compute nodes.

Only one job at a time is invoked by each PBS server on the cluster. The jmutex application will then ensure that only one of the jobs is actually executed. The PBS servers are not aware of that mutual exclusion, since it is done by the output interceptors. From the point of view of each of the PBS service processes, each runs the current job.

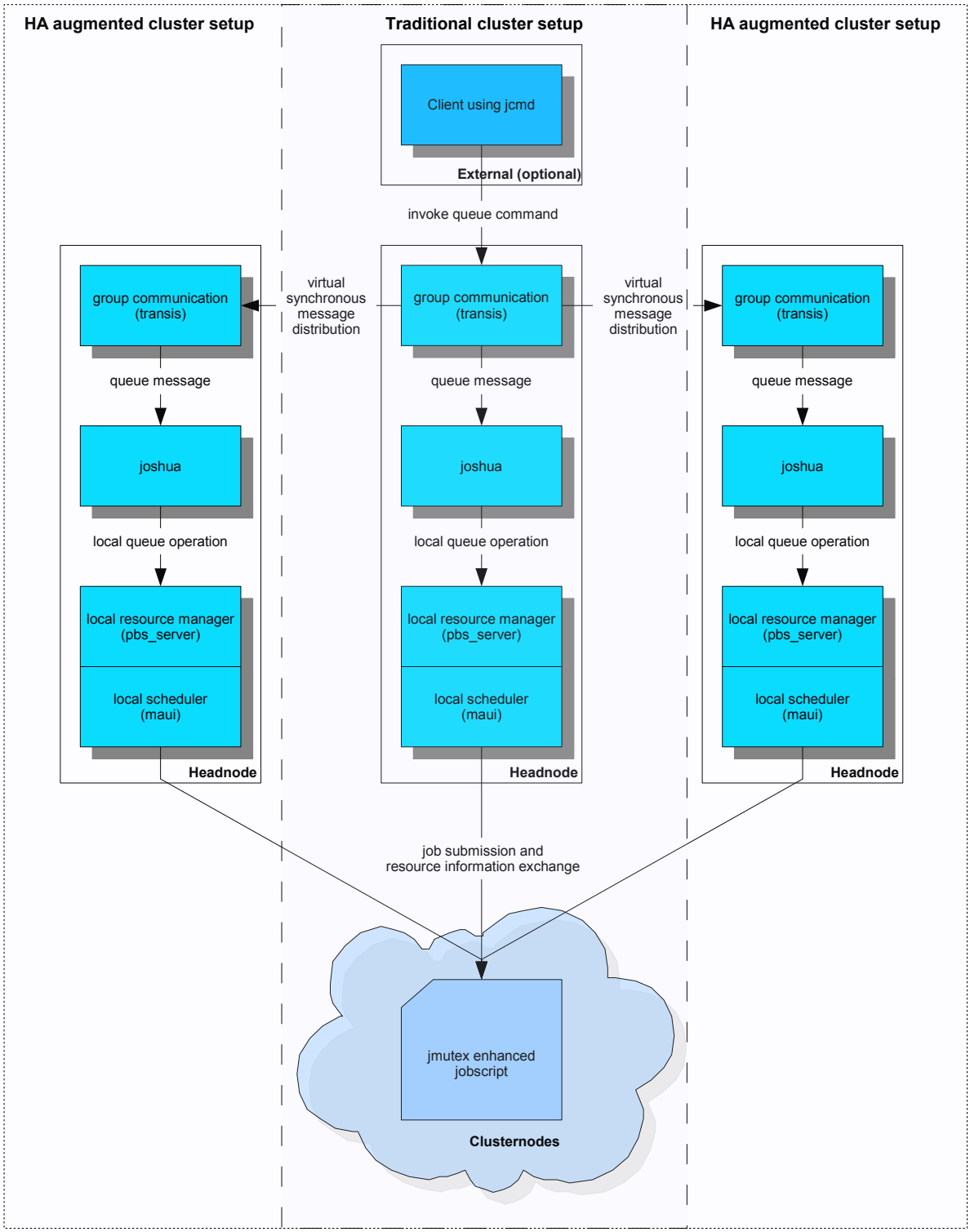

Figure 2.8.: JOSHUA components overview

2.1.6 Scalable availability

With the symmetric Active/Active high-availability model approach used for the job-scheduler service, multiple redundant active service components are possible. The huge advantage of using the proposed architecture for services is to have $1 + n$ head nodes with separate service processes on each node.

It is then possible to scale the availability of a target system to the availability needs of the site and the application to run on a cluster.

Since each head node operates in parallel to any other, system availability can be greatly improved. When all parts of the components fail and therefore are unable to provide the service, the virtual group of components is considered as operating in parallel. [Eve05b]

Such a combined system is operational if any component is available. It therefore can be concluded that the overall availability A of a symmetric Active/Active high-availability architecture with n components is $1 - (allpartsareunavailable)$. The following equation can be stated: [Eve05b]

$$A = 1 - (1 - A_{component})^n \tag{2.1}$$

As equation 2.1 implies, the combined availability of n components in parallel is always much higher that the availability of a single component. [Eve05b]

Now it is easy to combine as many components as necessary to scale the availability to achieve the needed reliability parameters. The most important parameters for a high-availability cluster system is the mean time between failure (MTBF) and mean time to repair (MTTR).

The Mean Time Between Failure is the average time between the failure of hardware modules. This estimated time is the average time before a failure occurs in a hardware component. Usually the MTBF for hardware modules or whole systems can be obtained from the vendor for off-the-shelf hardware modules. The Mean Time To Repair is the time needed to repair a failed hardware component. For an operational

system, repair usually means replacement of the failed hardware module. Thus the MTTR can also be viewed as the mean time to replace a failed hardware component. [Eve05a]

To demonstrate the new achieved scalable availability for the proposed symmetric Active/Active high availability job-scheduler service architecture, example availability and reliability calculation takes the following system values as basis: All head nodes are identical, with a $MTBF$ of $5000h$ and a $MTTR$ of $72h$. These values are taken from the experience of a system administrator for cluster systems.

The availability of a certain system or component can be calculated from the MTBF and MTTR as follows:

$$A_{component} = \frac{MTBF}{MTBF + MTTR} \tag{2.2}$$

The system availability is usually specified in a nines notation. 3-nines availability for example corresponds to 99.9% availability and a 5-nines availability corresponds to 99.999% availability. [Eve05a]

A more intuitive way to understand the availability of a system is to calculate the downtime per year from the availability percentage.

To calculate the downtime per year in hours, the following equation can be used:

$$t_{down} = 8760 \cdot (1 - A) \tag{2.3}$$

From the basic values of the exemplar head-node components, the following availabilities and downtime can be achieved by using different numbers of head nodes combined in parallel:

No. HN	Availability	Est. downtime
1	98,580441640%	5d 4h 21min
2	99,979848540%	1h 45min

Table 2.2.: Availability and downtime for different numbers of head nodes

No. HN	Availability	...continuation Est. downtime
3	99,999713938%	1min 30s
4	99,999995939%	1s
5	99,999999942%	18ms

Table 2.2.: Availability and downtime for different numbers of head nodes

This example shows that the Active/Active approach for high availability not only improves the MTBF, but also increases availability and reduces downtime of the system configuration enormously. A cluster head-node configuration of multiple node components can minimise downtime per year, which also translates to a significant cost saving. [She02]

2.2 System design overview

The proposed JOSHUA software system design outline proposes the following principles:

- use of the virtual synchrony paradigm
- global process state provided by group communication system transis
- external replication used for resource management and job scheduler
- messages are processed as uninterruptible events
- replaceable external components
- introduction and use of symmetric multi-headed cluster architecture
- number of head nodes arbitrary leading to scalable high availability

The following section shows how the proposed JOb Scheduler for High availability Using Active replication (JOSHUA) system is achieved by this Master thesis project.

Beside its general principles, JOSHUA proposes the following design guidelines: JOSHUA takes advantage of a commonly used variety of the Portable Batch System (PBS) [Alt03] PBS TORQUE as its basic component to evolve its system architecture derived from the traditional Beowulf cluster setup. As the use of PBS implies, active replication will be done externally due to the extraordinary amount of work needed for internal replication.

Taking advantage of the existing robust and broadly-used, productive, stable PBS reduces the effort needed to implement and introduce a job-scheduler service from scratch. Both, users and developers benefit from this. Users, especially non system engineers, will not have to get used to a new cluster software environment. In fact, the newly introduced system should be as transparent as possible, so that it appears that a commonly used Portable Batch System is running on the cluster.

The JOSHUA software system extends the existing capabilities of a PBS based system with high availability by creating a thin wrapper around the Portable Batch System.

The transis group communication system that supports efficient group multicast for high availability is the group communication system of choice for the implemented prototype. The system was developed in the high-availability laboratory at the Computer Science Department of the Hebrew University of Jerusalem. [The99b]

2.2.1 JOSHUA server daemon

The JOSHUA server daemon is the main part of the JOSHUA cluster system. It acts as a transparent gateway to hide the newly introduced high availability of the underlying PBS TORQUE from the user, and is thereby also the input interceptor for the locally-running resource management and job scheduler processes proposed as the design of Active/Active high availability using external replication.

The server application uses the transis group communication system to share all process states and events among the participating daemons, as figure 2.8 illustrates.

Transis can be seen as a multicast group communication layer that facilitates the development of fault-tolerant, distributed applications in the cluster network environ-

ment.

As a matter of fact, transis supports reliable group communication for high-availability applications and is, therefore, the preferred service application of choice for this software project. It also contains a novel protocol for reliable message delivery that optimises the performance of existing network hardware. [The99b]

The transis group communication system can cover all possible events that occur during the membership of an augmented head-node setup with JOSHUA within the formed group. Every JOSHUA server (*joshua*) is therefore able to become aware of every group membership change, from joining of new members, as well as failing or exiting members.

The provided communication facilities of transis are used by *joshua* to exchange information from and to the client command tools and the mutual exclusion components running outside the head node. Fortunately *joshua* does not need to take care of the distribution of messages, but reacts appropriately to incoming messages.

The JOSHUA server daemon must handle message events like:

- new daemon requests to join the group
- failing or exiting group members
- the failure of an external component of the JOSHUA server
- job is submitted by a client
- job is deleted by a client
- client requests job status information
- requests for job execution
- job is finished

2.2.2 JOSHUA user commands

The command line user tools are the facilities JOSHUA offers to allow its users to perform queue operations, such as:

- job submission
- job status information
- job deletion

All existing PBS commands for these capabilities will be replaced by a JOSHUA alternative. Since one goal is to achieve highest transparency to the user, the replacements act exactly like the originals.

To meet these requirements, the commands use the same input and option switches as the originals. Since the input of the tools is not processed on the client side, all program parameters will be seamlessly redirected to the JOSHUA server daemon to feed the input of the external resource manager and job scheduler.

As figure 2.8 shows, the *joshua* daemon locally invokes the requested user command on each head node. There, the server application takes advantage of the original PBS processing tools. Thus, their output is redirected to the *jcmd* user tools, the proposed grade transparency can be guaranteed, even if the underlying Portable Batch System changes.

2.2.3 JOSHUA cluster mutex

The cluster mutual exclusion *jmutex* is the JOSHUA component that solves the single-instance-execution problem. Since the participating *joshua* daemons of the job scheduler group on the head nodes run in virtual synchrony, each of the resource managers holds the same queue and tries to run identical jobs concurrently. Any duplicate execution must be prevented by proper counter measures that intercept the output of each head node as proposed by the design of the Active/Active high availability model using external replication.

By technical programming measures, a concurrently invoked job represents a critical section of a non-shareable resource between concurrent processes, in this case job schedulers. A mutual exclusion algorithm (also known as mutex) prevents the processes from entering that critical section.

Unfortunately, a job might be invoked by a PBS TORQUE server on any PBS mom on the cluster. JOSHUA solves this problem by using a distributed mutual exclusion algorithm for single-instance execution.

Fortunately, the virtual synchrony provided by the transis group communication system can be used to solve that problem, too. Since all the members of the *joshua* daemon run in virtual synchrony, all messages are received by all processes in the same total order.

Jmutex takes advantage of that by releasing a message to the JOSHUA daemon before invoking the job. Any message by the first executor receives permission to actually execute the job.

All other executors placing that request will be suspended. As soon as the first executor has finished the job successfully, a release message is invoked, which wakes up all pending executors. Since the current job is thereby done, all pending executors will be ordered to carry on with the next job in the queue.

In order to get this design work, it is necessary that the underlying Portable Batch System is set up in a dedicated mode, where each job gets exclusive access to all the available compute nodes.

3 Implementation Strategy

Although this Master thesis project only develops a proof-of-concept software, some common standards for open source software should be met by the implementation to improve further development and make reuse of code and results even possible.

The implementation approach follows the following guideline where possible:

- use of POSIX conform libraries and library functions
- abdication of proprietary und non-portable code of libraries
- primary target hardware architecture: x86 32-Bit Intel Architecture (IA32)
- primary target software platform: GNU/Linux
- programming language: C
- complete code and API documentation using doxygen [vH05]
- make use of GNU Autotools [Fre05] for:
  - easy configuration on target system
  - resolving necessary software dependencies
  - package preparation for installation and deployment
- reuse of common open source libraries
- avoid reinvention of already existing code

In order to reduce similar code for components of the JOSHUA system, the implementation will be split into logical parts, which reuse identical modules via an internal library. The main parts for the implementation are therefore:

- JOSHUA server application (*joshua*)
- JOSHUA cluster mutex (*jmutex*)
- JOSHUA client tools (*jcmd*)
- JOSHUA utility library (*jutils*)

The utility library integrates the sum of all shared functions and capabilities used by the main JOSHUA components. The *jutils* contain facilities for:

- double linked lists
- logging
- message creation and recognition
- input and output
- miscellaneous

Besides the internal utility library, external components like the transis communication library *libtransis* and the configuration file parser library *libconfuse* [Hed05] need to be integrated for the implementation of JOSHUA as well.

Figure 3.1 outlines the different internal components to implement and the external library dependencies to integrate.

Other external software components are needed during runtime as well, such as:

- transis group communication daemon
- runtime library of *libconfuse*
- client tools of resource management system (such as *qsub* from PBS TORQUE)

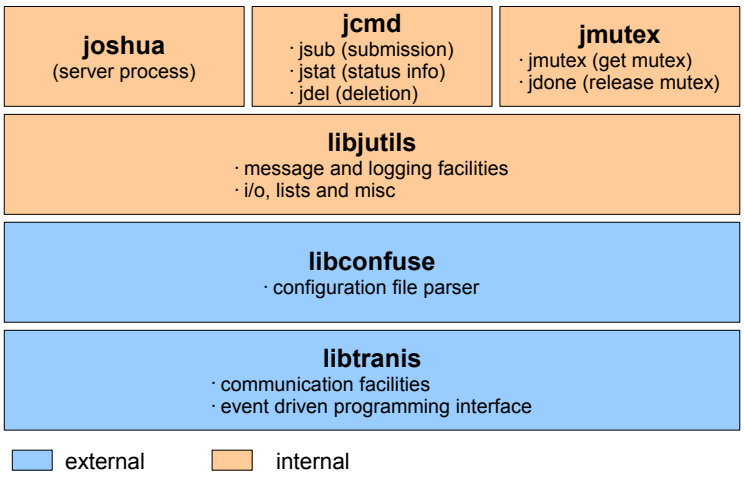

Figure 3.1.: JOSHUA component implementation and integration overview

- resource management server daemon (PSB TORQUE server)
- job scheduler (Maui Cluster Scheduler^{TM})

3.1 System implementation approach

3.1.1 JOSHUA server daemon

Internal data

The JOSHUA server is the most important part of the implementation as it acts as the local interceptor for all client requests made globally to all redundant running JOSHUA daemon in the job-scheduler service group.

As the server daemon supports dynamic changes of the group such as a head node joins or fails, the necessary data for the status snapshot replication must be stored by internal data structures. In order to eventually share the current process state with another joining group, the JOSHUA server daemon stores the following relevant process information needed for a snapshot:

- job submission messages for queued jobs greater that the last finished job

- job delete messages for job identifiers greater than the last finished job
- information about currently running jobs
- information about currently finished jobs
- job identifier of the last successfully submitted job to the local resource management system
- job identifier of the last successfully finished job
- generation counter for submitted jobs
- generation counter for finished jobs

As the list of necessary data for the snapshot replication implies, the data is needed to give the joining JOSHUA server all data concerning the state of the job scheduler and resource management process to intercept.

The job submission and deletion messages as well as the information about currently running and finished messages are implemented using a dynamically linked list to give the server process the flexibility and scalability to manage as much jobs as possible with the underlying PBS TORQUE.

By using the job queue information from the existing process, the new JOSHUA process can locally replicate the exact same state of the service process That way it is ensured, that the new process is eventually in virtual synchrony with the other interceptors.

In order to minimise the necessary data to store and replicate for the joining event, only job submission and deletion messages of yet unfinished jobs will be stored and used for the queue replay. In order to get the same job numeration on the local job scheduler and resource management service, the job identifier of the last finished job is used as start-up value for the PBS TORQUE server. That way it is ensured that all queues hold the same jobs with identical identifiers within the job scheduler group.

First all submission messages will be locally replayed, followed by the submission messages in order to ensure that the "gaps" in the queues are replicated, too.

The data about the job identifier generation is needed to successfully decide by the cluster mutex, which jobs are allowed to be placed into execution on the cluster. The used PBS TORQUE has a turns its job identifiers over every 99999999 jobs, given by the *PBS_SEQNUMTOP* number. That means after 99999999 submitted jobs, the next assigned identifier will be smaller than the last one again. In order to prevent execution rejection, a turnover can be recognised by JOSHUA using the generation counter for both submitted and finished jobs.

Server events

In order to ensure the virtual synchrony among the replicated JOSHUA server processes, the daemon is implemented by using event-driven programming. Since every message is globally ordered delivered by the transis group communication daemon and each message will be processed in an uninterruptible event in the delivered order, the virtual synchrony can be easily achieved by the implementation of message triggered events.

As the central interceptor point for the externally replicated job and resource management system, the JOSHUA server daemon has to handle several messages can trigger events, such as:

- group membership change messages:
  - a new daemon request to join the group
  - a member fails or leaves the group
- group communication messages:
  - job submission messages
  - job deletion messages
  - queue status information request message
  - execution request messages of jobs on the cluster nodes
  - execution finishes messages from the cluster nodes

The facilities for the event-based programming are provided by the transis group communication library and will be described in further detail in 3.2.4.

Group membership events

As already mentioned in 3.1.1, each JOSHUA server process keeps an internal list of important data for a joining process in order to replicate the current process state.

In order to achieve an identical state between the joining and the assisting process a join event takes place for the both of them. Since events cannot be interrupted and all pending messages will not be processed before the join event is executed properly. That way the virtual synchrony is ensured, because after a join both existing and new process are identical and can catch up to other running service members by processing the remaining messages.

The join event is implemented by using the group communication facilities. Since the assisting process only needs to replicate its current status to the new one, it basically dumps all internal data to the joining process and carries on. This asynchrony is possible, because the new member first processes all snapshot replication messages, before it becomes an established member of the service group.

The state diagram in figure 3.2 illustrates the join event. After both processes have finished the event, they have either exchanged all internal stored data and the new member has successfully joined or the join process fails. In case of any error or failure during the join event, the new member leaves the group and shuts down.

In contrast to the join event, in case of failure of a member process, the group only internally updates its member list. Since all remaining service processes share the same state, they can carry on without any further operation.

Group communication events

Usually messages to process for the JOSHUA server are communication messages intercepted from the client tools. Most of those messages from the clients invoke a queue alteration command locally executed by each of the service processes. The

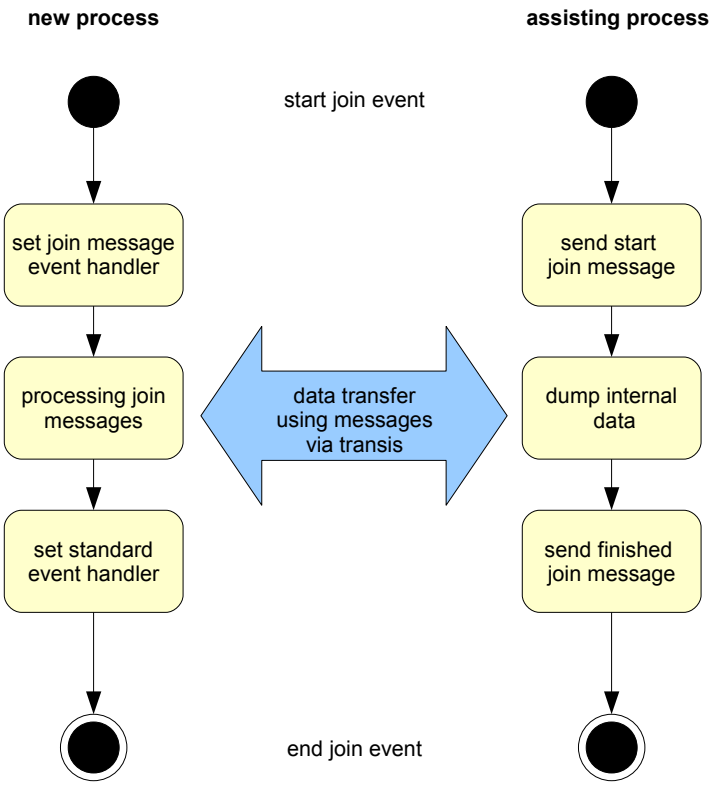

Figure 3.2.: State chart diagram join event

implementation approach of the user commands can be found in further details in section 3.1.2 and figure 3.3.

Each user message is uninterruptible processed within an event, ass well. With every message causing an event, the further handling is decided by the information content. An incoming message triggers an event and the handler decides upon the header, which input has to be generated for the external job scheduler and resource management service process by executing a proper command.

The client side of any queue command gathers all necessary information, like the argument vector, the standard input, the current working directory and environment to send it packed in a transis message to one of the head nodes. The transis daemon takes care, that the message reaches all group members.

The program flow chart in figure 3.3 shows, how the JOSHUA users commands interact with the server process, which produces the local input for the external job

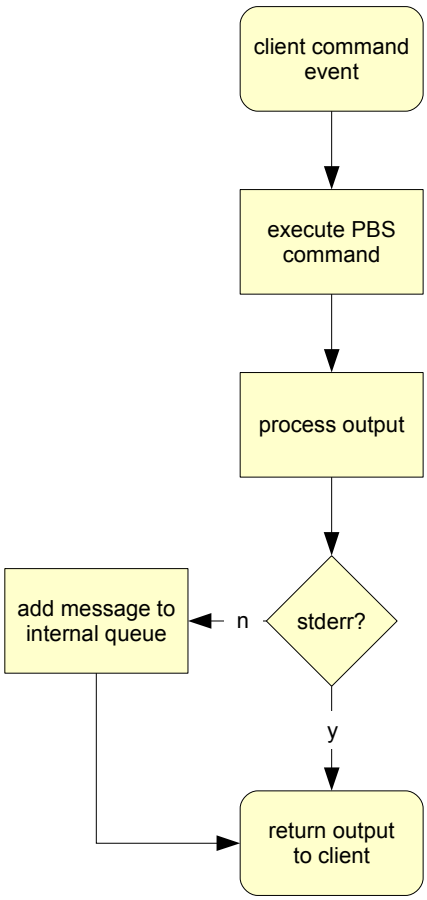

Figure 3.3.: Flow chart client command event

scheduler and resource management service process. On the server side the incoming message is recognised as an event and the appropriate PBS TORQUE command is invoked locally using the previously gathered execution environment data from the client. Standard output and error from the PBS command are collected and send back to the client application, providing the proposed transparency to the user. Restrictions concerning that include that the interactive mode for job submission is not possible and the message size is limited by the transis communication facilities.

3.1.2 JOSHUA user commands

As shown in section 3.1.1, the JOSHUA user commands are used to generate the local input for the PBS TORQUE by distributing a request through transis to all JOSHUA

servers in virtual synchrony.

On the client side the following PBS commands are replaced by their appropriate JOSHUA pendants:

- *qsub* → *jsub*
- *qdel* → *jdel*
- *qstat* → *jstat*

The new "j" commands can be transparently installed on the client side using alias, symlinks or simple renaming on a common UNIX system.

All messages from the clients follow certain encapsulation rules in order to be recognised correctly by JOSHUA. Since the effective user command is executed on the server side, only the environment values of the execution are used to mask the local invocation.

In order to transparently shift the execution to the server side the standard input, the argument vector and the environment are captured and encapsulated into a transis message and send to the JOSHUA server as illustrated in figure 3.4.

The command output is generated from the response message of one of the remote JOSHUA server applications, as shown in section 3.1.1.

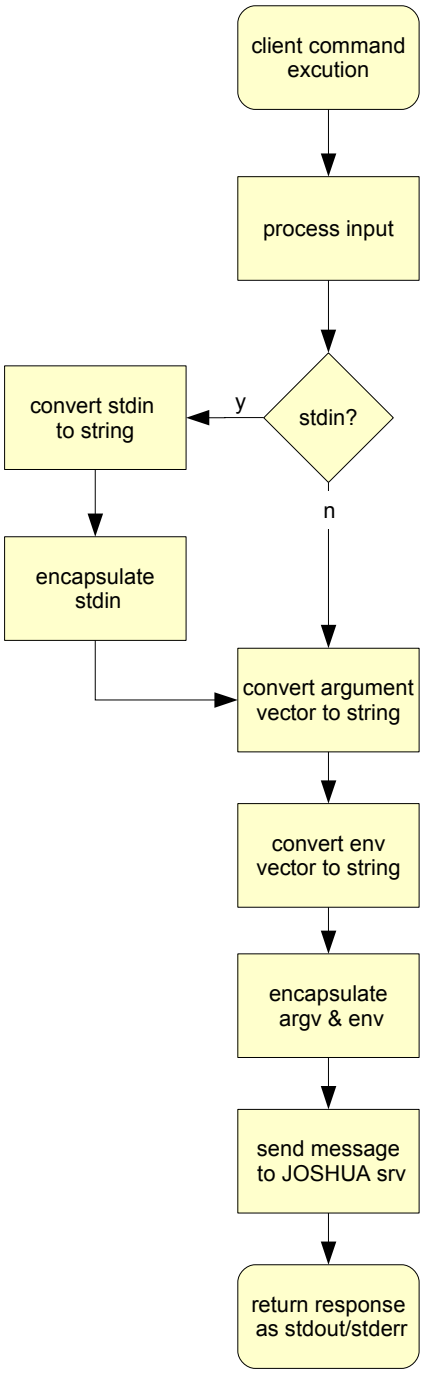

Figure 3.4.: Flow chart client command execution

3.1.3 JOSHUA cluster mutex

The cluster mutex implementation of JOSHUA (*jmutex*) represents the output interceptor from the distributed external job scheduler and resource management service processes. Since all of these service processes try to execute the current job, the cluster mutex ensures that the output, in this case the execution only takes place once.

The implementation of the mutex has to take advantage of the anyway used group communication service. Only that way it is possible to make a mutual exclusion distributed. Since all messages to the JOSHUA servers are virtual synchronous and therefore globally ordered, the request to execute a job can be recognised by all service processes equally.

The used TORQUE version provides the ability to run prologue and epilogue scripts before and/or after each job is executed. With such a script, usually a site can prepare systems to perform node health checks and similar things. [Clu05b]

For the implementation of *jmutex*, the prologue and epilogue scripts are used to execute transis communication stubs. By design of the scripts both prologue and epilogue have access to the job id of the current job to execute and can send it to the JOSHUA server for verification.

The prologue script only allows the first execution request to actually invoke the job, by getting a positive response message to return from the prologue script by the JOSHUA server and from there carry on with the execution. All other job execution requests with the same job identifier are put to halt by being locked in the stub until released by the service group that the job is already done.

When the execution of the current job is finished, the epilogue script is invoked, which produces a "job finished" message, to let the JOSHUA server daemons release the pending job execution requests, so that next job can be placed in execution eventually.

3.2 Integration of external components

3.2.1 Runtime dependencies

Since the JOSHUA service system uses external replication, its proper operation depends on external processes. Therefore the JOSHUA server daemon forms a special group with its external processes. This group consists of:

- JOSHUA server application (*joshua*)
- job scheduler (for example Maui Cluster Scheduler)
- resource management server daemon (e.g. PSB TORQUE server)
- transis group communication daemon

Only if all these processes are running, the system can work as proposed. In order to ensure the safe operation of the process group a mechanism has been implemented to observe the status of each process group member.

If one of the process group members fails, all remaining parts must be shutdown. Therefore tow dedicated processes start-up and observe the participating processes. This start-up sequence is realised by *jinit*, as shown in figure 3.5 and also takes care of the correct initialisation of each participating process during the start-up.

That way the *jinit* process cannot only recognise a failure of one of the processes, but also starts the processes in a required order as shown in 3.5.

The startup sequence is implemented by using a configuration file parsed by the runtime library of *libconfuse*. The processes to start are ordered and named as follows:

1. *group_com = "transis"*
2. *scheduler = "maui"*
3. *joshua = "joshua"*
4. *job_server = "pbs_server"*

Besides the process names, for each process the path for the executable and the configuration file is given, to get the processes started properly by *jinit*. An example of a complete configuration file can be found in the Appendix in A.1.3.

Usually the first process to start is the group communication service, since it is vital part of all communication. Fortunately the implement ion uses the transis group communication daemon, which status is permanently checked by *joshua* itself in connection with transis communication library *libtransis*. Without transis, the JOSHUA server application immediately fails and would not even start up. Therefore the configuration file includes the *group_com* entry just for portability issues (see refport for further details).

Next to start is the *maui* scheduler, followed by the *joshua*. The *jinit* process waits to start the resource management server daemon of PSB TORQUE until *joshua* can join the head-node group to obtain the job identifier for the first PBS job during the first stages of the join process (see 3.1.1).

When the job identifier has been set by *joshua* in the PBS server database, it signals *jinit* to continue to start the *pbs_server*. The implementation uses the signals *SIGUSR1* and *SIGUSR2* for process notification. As soon, as the resource manager is up and running, the join process of *joshua* can continue and jobs can be submitted and replicated to the queue as needed.

In order to observe the status of each process, *jinit* forks and executes all the members of the process list for the start-up. Since *jinit* is the father process of each forked process, it will be signalled by the *SIGCHLD* by the operating system, when on of its children gets killed of fails.

Since *jinit* started all child processes, it also knows their process ids (PID). Thus, if one of children stops working, *jinit* can send a *SIGKILL* to ensure to all its remaining children, which ensures that remaining parts of the JOSHUA components can interfere with other JOSHUA service processes, since the proper operation can not be guaranteed, when parts fail.

In order to ensure the detection of a failure of *jinit* it creates a clone of itself, which observes the original *jinit* using a pipe. When the first *jinit* fails or gets killed, the

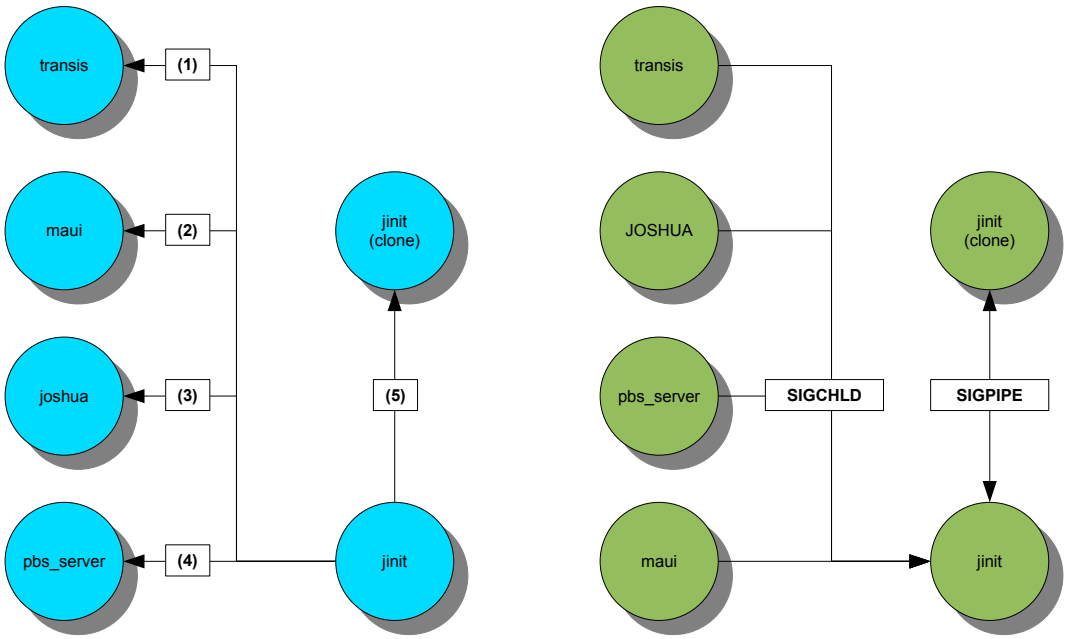

Figure 3.5.: Start order and signals for cooperation with external components

pipe breaks and a the clone gets a *SIGPIPE*. In this case the clone is responsible to shut down all remaining processes of the group. ´ In fact, this implementation of a self-observing process group relies on the support of signals by the operating system. This is supported by the primary target software platform is GNU/Linux, although signals are also supported by most Unix-like systems.

3.2.2 Group communication system

The transis group communication system supports process group communication using a communication library to implement own application using the communication facilities.

Those groups are conveniently identified by a name that can be chosen by the user. That way messages can be addressed to an entire group only by specifying the name of the group. [Mal94]

Using this group abstraction, the communication subsystem relieves the user from identifying the targets of messages explicitly, and from finding the network routes to

them. In addition, it guarantees all-or-none delivery semantics, and handles message losses and transient network failures transparently to the application. [Mal94]

The transis group communication system supports the following forms of group multicast operations:

- FIFO ordered
- causally ordered
- totally ordered
- safely ordered

In order to achieve the necessary virtual synchrony for the intended operation of JOSHUA, only the safely ordered communication operation is used.

The safe mode guarantees the delivery of message after all participating group members in the network have received a copy of that message. This mode not only assures total order, but also that all members receive the exact same message. [Mal94]

3.2.3 Communication facilities

The group communication facilities provided by the transis group communication system are relatively easy to use and implement in customised application, which want to take advantage of the communication capabilities of a communication subsystem.

In order to use any transis communication facility the application developer just has to include the group communication header file *zzz_layer.h* and link the program to the transis communication library (libtransis.a). [Mal94]

All communication capabilities of transis rely on the transis group communication daemon. Each client using the communication facilities must connect to that daemon.

The connection can by established either be locally to a local running transis daemon or remotely to a transis daemon running on another machine with in the network.

Either way, for communication a message box has to be created, in order to receive any message.

By using the *join* function it is possible to join a certain group, such as each *joshua* is member of the head-node group.

All messages addressed to that group or directly to the client will be delivered to the previously created message box. All messages will be stored on the message box until explicitly fetch by the receive function. The usual receive is implemented as a blocking function, which only return, when a message could be fetched from the message box. [Mal94]

The provided send functions by transis can set the used group multicast operation mode and the addressed communication group explicitly for each send operation. [Mal94]

All messages within the group communication system are restricted to a length of *MAX_MSG_SIZE* (64kByte). A full API reference can be found at `ftp://ftp.cs.huji.ac.il/users/transis/tutorial.ps.gz` [Mal94]

The following basic example shows how to create a message box, join a group (*HEADNODEGROUP*) and send and receive a message:

```
1  /* open transis conection */
   msgbox = zzz_Connect(coname, stack, flag);
3
   /* join the headnode group */
5  zzz_Join(msgbox, HEADNODEGROUP);

7  /* send a message the headnode group */
   amount = zzz_VaSend(msgbox, SAFE, 0, strlen(send_buf)+1 , send_buf, HEADNODEGROUP, NULL);
9
   /* receive a message */
11 amount = zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
```

3.2.4 Event-driven message operation

As recommended by the transis group communication user tutorial, the very nature of the system design of the JOSHUA server daemon requires a special programming technique in order to handle all group communication as desired. The most efficient way to handle such a message driven system, is by using event-driven programming.

Fortunately the transis communication library already provides event facilities. In general a message triggered event system works similar to signal handling under UNIX. In this case the communication application runs in an endless loop waiting for pending messages to trigger events. These events will eventually handles using previously registered event handlers. [Mal94]

The advantage of the provided event-driven message operation families are a better program structure of the resulting code and that each of the events is non-interruptible. That way all incoming messages will be handled sequentially as intended by the operation in virtual synchrony.

When for example a new member invokes a join event, this event cannot be interrupted by any other event. That way the handling of such an event will always end in a previously defined way, such as either the new member joined successfully or not.

After an event is handled, other pending messages can be processed. Due to the virtual synchrony all events occur on all participating group members in the same total order. [Mal94]

The following snippet shows, how an event handler for incoming messages can be registered, before the program control is given to the transis library by invoking *E_main_loop()*:

```
1 /* add event handler for incoming messages */
  zzz_Add_Upcall(msgbox, eventhandler, USER_PRIORITY, (void *) 1);
3
  /* start event handler */
5 E_main_loop();
```

All JOSHUA server applications operate in this message triggered loop to react appropriately to incoming group communication and membership event messages (see 3.1.1 for details).

3.2.5 Resource management system and job scheduler

The supported resource management system by JOSUA is PSB TORQUE since version 2.0p1. In order to work properly with the *jinit* startup routine and process obser-

vation, the default behaviour of the PBS server has to be changed.

By default a starting PBS TORQUE daemon forks itself on start-up and its process status can therefore cannot be observed by *jinit*. The fork can be disabled by running the PBS server process in a light debug mode, which does not significantly influence the daemons operation.

In order to built a compatible version of PSB TORQUE for the JOSHUA system, it must be compiled and configures with the *–multi-server* option, which enables the support of multiple PBS server and moms (see 2.1.5).

The same forking problem also occurs with the Maui Cluster Scheduler^{TM}and can also be solved by running in debug mode.

The job scheduler has to be configured in a certain way to work with the JOSHUA system design. By setting the value *NODEACCESSPOLICY* to *SINGLETASK* in the Maui configuration file, the job scheduler will give each running job exclusive access to all cluster nodes. This behaviour is needed in conjunction to the implementation of the JOSHUA cluster mutex.

3.3 System tests

The process of software tests is needed to identify the correctness, completeness and quality of the developed software implantation. Testing is nothing more but criticism and comparison toward comparing the actual values of program with expected ones. [Wik06b]

These expectations directly derive from the software system requirements and milestones. The completeness of the implementation can be measured by how good the final version meets the requirements. The tests include, as well as the requirements, definition of certain capabilities and responses behaviour by the software system. This behaviour in reaction to the probing of the tester will be evaluated.

The quality of the tested software can vary widely from system to system. Therefore some of the common test techniques include reliability, stability, portability, main-

tainability and usability issues as well, besides the meeting the software development requirements. [Wik06b]

In general, software faults and software failures are distinguished. In that case, a failure means that the software does not do what the user expects of the requirements dictate. In case of a fault, a programming error may or may not manifest as a failure (e.g. memory allocation error, which do not directly lead to a segmentation fault). That way, a fault is an indicator for the correctness of the semantic of a computer program. [Wik06b]

Nevertheless, regardless of the uses techniques, the desired result of testing is a level of confidence in the software system. That way, a developer is confident that the software has an acceptable defect rate. The acceptance of that defect rate depends on the nature of the software. [Wik06b]

Since this Master Thesis projects developed a software system to increase high availability, special code correctness of the implementation is of vital importance. Since the core component of the symmetric Active/Active job-scheduler service is the JOSHUA server daemon, especially the memory tests for memory leaks and segmentation faults are of major concern.

Disregarding the common practice of software testing usually performed by an independent testing group, the tests for the JOSHUA application components are done by the developer. [Wik06b]

This is acceptable, because the developed software is only proof-of-concept. Besides, a separate testing group would only lay to project delays, which cannot be afforded by a Master thesis.

Hence, software testing begins when the project starts, that way it progresses as a continuous process until the project finishes. This both ensures the final version more likely to meet all requirements and also helps to keep the software project in schedule. It also ensures that defects can be found earlier and therefore are easier to fix. [Wik06b]

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness. [Wik06b]

There are many approaches and techniques for testing software. In order to assure the quality of the JOSHUA software system the following test techniques will be used to evaluate the implementation:

- stress and performance test
- memory allocation test
- system test

All test are performed in a dedicated cluster environment setup for the development and tests of the JOSHUA software system. Each cluster and head node has the following properties:

- Hardware

  CPU Dual Intel Pentium III (Katmai) with 450MHz

  network Fast Ethernet 100MBit/s full duplex

- Software

  operating system Debian GNU/Linux 3.1 (sarge)

  C compiler gcc version 3.3.5 (Debian 1:3.3.5-13)

  transis daemon and library version 1.03

  libconfuse library version 2.5-1

  resource manager PBS TORQUE version 2.0p5

  job scheduler Maui Cluster Scheduler^{TM}version 3.2.6p13

3.3.1 Stress and performance test

Stress tests are used to determine the stability of a developed software system. This kind of test may push the application beyond the capacity of normal operation, sometimes up to a breaking point, in order to see how the program behaves. [Wik06b]

In general that kind of testing refers to a practice of modelling the expected use of a software program by simulating multiple users accessing the program services concurrently. Therefore this testing technique is more relevant for a multi-user system, like a client/server model, such as web servers. [Wik06a]

The JOSHUA software system can be tested that way as well. For example, by forcing the program to process an unusual huge amount of input data, such a heavy job submission request by pushing it to the limits of input handling.

When such a test is placed on the system beyond normal use patterns, the responsiveness and performance of the system at unusually high or peak loads can be tested and evaluated. The load is that usually great that some error conditions are part of the expected results. [Wik06a]

The specific goals of the designed stress and performance test applied to JOSHUA are to evaluate the reliability and behaviour of the proof-of-concept software system especially in connection to the group communication subsystem.

The performance evaluation should demonstrate that the JOSHUA system meets performance criteria. This particular means that the tradeoffs for using an underlying group communication system to achieve new high-availability capabilities for a job-scheduler service is within a reasonable scale.

The stress and performance test used, stresses the JOSHUA system by constantly submitting several amounts of jobs to the cluster management system. The test will show how the system performs under the heavy load.

The same test is also applied to a traditional single-headed PBS TORQUE cluster setup in order to evaluate the tradeoffs and performance differences, when using the proposed JOSHUA cluster framework.

Under stress, the JOSHUA system sometimes suffers from deficiencies caused by the transis group communication system. Obviously this system was not built to sustain that heavy load from user input.

For performance tests, several system setups have been tested, from the common single-headed PBS TORQUE cluster to up to quad-headed JOSHUA setups. The total

submission time for jobs did not make much of a difference between a single-headed and multiple-headed JOSHUA.

As expected the times for the job submission where greater when using JOSHUA in comparison to the common PBS TORQUE architecture due to the communication overhead created by the group communication system.

With the stress and performance test is possible to examine, how the underlying new HA capabilities effect the user. The submission performance test shows, how the communication overhead caused by the group communication subsystem impacts the time for job submission.

As the test results in table 3.1 show, the tradeoffs for high availability is indeed up to three times the amount of time of a common PBS TOQRUE setup for the average submission, but not unreasonable. Though, the average submission time of one message with PBS is increased from 0.098*s* to a maximum of 0.349 for four head nodes, the user impact is minimal in practice, since those times are only fractions of a second.

System	**Head nodes**	**10 jobs**	**50 jobs**	**100 jobs**	**Average per job**
PBS TORQUE	1	0.93s	4.95s	10.18s	0.098s
JOSHUA	1	1.32s	6.48s	14.08s	0.134s
JOSHUA	2	2.68s	13.09s	26.37s	0.265s
JOSHUA	3	2.93s	15.91s	30.03s	0.304s
JOSHUA	4	3.62s	17.65s	33.32s	0.349s

Table 3.1.: Job submission performance test results

More importantly as the graph 3.6 shows, the submission times always increase linear with the number of submitted jobs. Though, the increase of almost 25*s* in submission time for 100 jobs in a four head-node setup seems high but it is not of practical relevance on a cluster system.

The graph also shows that the submission time depends from the number of running head nodes. The communication overhead increases for more than one head node

observably, because even when the JOSHUA components run on the same node, the submission is always done using the communication facilities provided by the transis group communication system.

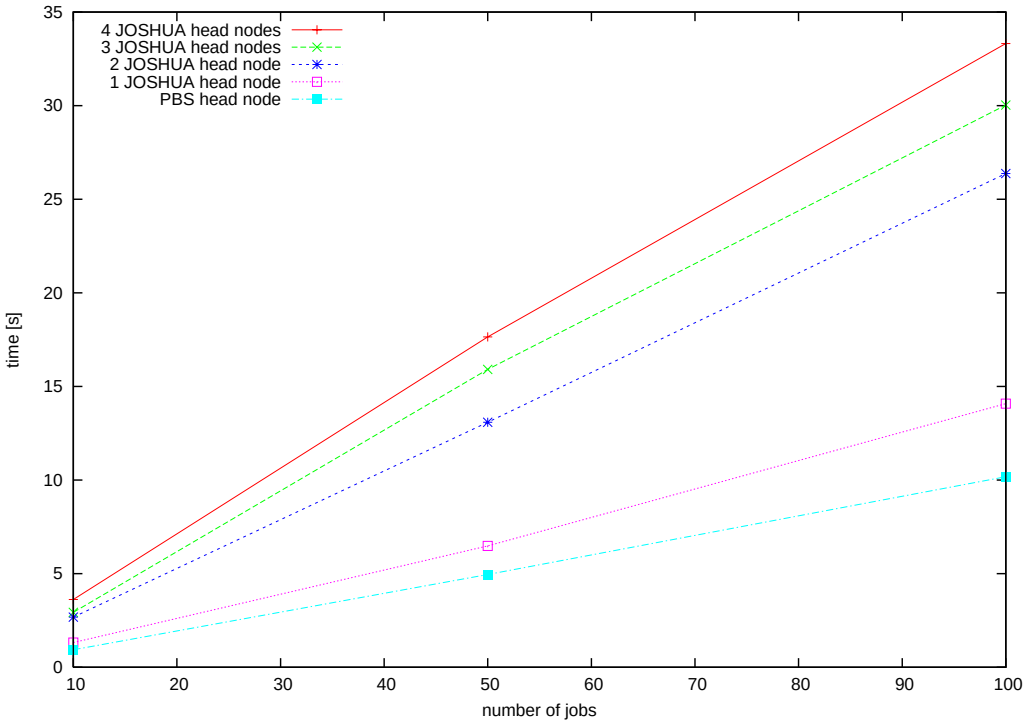

Figure 3.6.: Job submission performance tests

3.3.2 Memory allocation test

A memory allocation test is applied to the JOSHUA server daemon in order to assure a memory safe programming. As the daemon usually would have to run for a long amount of time with in a high-availability environment, the test should prove, that the server application will not malfunction under normal system load due to memory allocation errors, leading to buffer overflows, memory leaks or segmentation faults.

The allocation test is done by using the memory debugger *valgrind* [Val06]. Valgrind is famous suite of tools for debugging and profiling of Linux programs. By using those tools that come with Valgrind, one can detect many memory management bugs automatically. It eventually helps to avoid hours of bug tracing, makes programs

more stable and helps to speed up and reduce memory use of an application. [Val06]

As the output in appendix A.2.1 shows, the only possible memory leaks are caused by the tranis group communication library interface. For the memory allocation test, the joshua server had to safely receive and execute about 100 job submissions.

3.3.3 System test

According to the IEEE Standard, a system test is conducted on a complete, integrated system to evaluate the compliance of the system with previously specified requirements. In general Alpha and Beta testing of the applicable milestones are subcategories of system tests. [Wik06b]

System testing takes all of the software components that have successfully been developed and integrated to evaluate if the whole system works as expected. Those tests include the software system itself integrated with the applicable hardware system. [Wik06b]

In general, for a system test, the entire complex system can be tested as an integrated whole against the requirement specifications for the first time. [Wik06c]

System testing tends to be more of an investigatory testing phase, where the focus is not only the design, but also the behaviour and even the believed expectations toward the eventual developed solution. That way, you can see the system test as the final testing phase before acceptance testing. [Wik06c]

The following examples are different types of testing, within System testing, that should be considered before System testing begins: [Wik06c]

- functional testing
- user interface testing
- error exit testing
- performance testing
- sanity testing

- reliability testing
- recovery testing

The following specific system capabilities have been tested on the integrated JOSHUA software system:

capability	expected behavior	command	output verification	pass
		job submission		
submit job with valid jobscript	return job identifier	*jsub script.sh*	`0.h1.mycluster.org`	✓
submit job with invalid jobscript	redirect error message from PBS TORQUE	*jsub invalid.sh*	`qsub: script file:: No such file or directory`	✓
submit job via STDIN	return job identifier	*echo "hostname" | jsub*	`1.h1.mycluster.org`	✓
submit job without transis running	return error message	*jsub script.sh*	`Error: RemoteConnect to host h2.mycluster.org connection failed.`	✓
submit job without joshua running	return error message	*jsub script.sh*	`Error: Operation timed out.`	✓
submit job with illegal switch	return error message	*jsub -h script.sh*	`Error: Option not supported.`	✓
		job deletion		
delete job by valid job identifier	return nothing	*jdel 1*	n/a	✓
delete multiple jobs with valid job identifier	return error message	*jdel 1 2 3*	`Error: Multiple job deletion or option not supported.`	✓
delete job with invalid job identifier	redirect error message from PBS TORQUE	*jdel 5*	`qdel: Unknown Job Id 5.h1.mycluster.org`	✓

Table 3.2.: System test JOSHUA command line tools

capability	expected behavior	command	output verification	pass
			...continuation	
delete job without transis running	return error message	*jdel 1*	`Error: RemoteConnect to host h2.mycluster.org connection failed.`	✓
delete job without joshua running	return error message	*jdel 1*	`Error: Operation timed out.`	✓
delete job with illegal switch	return error message	*jdel -W 10 1*	`Error: Multiple job deletion or option not supported.`	✓
job queue status				
get queue status when queue filled	redirect output from PBS TORQUE	*jstat*	`Job id Name User Time Use S Queue ---------- -------- -------- ---- - --- 1.h1 test.sh kai 0 R batch`	✓
get queue status when queue not filled	redirect output from PBS TORQUE (empty)	*jstat*	n/a	✓
get queue status without transis running	return error message	*jstat*	`Error: RemoteConnect to host h2.mycluster.org connection failed.`	✓
get queue status without joshua running	return error message	*jstat*	`Error: Operation timed out.`	✓
initiate server command				

Table 3.2.: System test JOSHUA command line tools

capability	expected behavior	command	output verification ...continuation	pass
startup server with valid config file	startup joshua and external components	*jinit -c config*	`jinit started... Attemting to start JOSHUA components... job scheduler..... done joshua..... done resource manager..... done jobserver..... done see logfile in Appendix A.2.2`	✓
startup server without transis	startup joshua and external components	*jinit -c config*	`jinit started... Attemting to start JOSHUA components... job scheduler..... done joshua..... failed. Check logs for further information.`	✓
startup server without config file	return error message	*jinit*	`Error: Missing config file. [...]`	✓
startup server with invalid config file	return error message	*jinit -c /tmp/joshua.conf*	`Error: /tmp/joshua.conf:1: no such option 'headnoodes'`	✓

Table 3.2.: System test JOSHUA command line tools

capability	expected behavior	output verification (shortened)	pass
job submission message received	submit job locally and redirect output to client	see logfile in Appendix A.2.2	✓

Table 3.3.: System test JOSHUA server

capability	expected behavior	output verification (shortened)	...continuation pass
job deletion message received	delete job locally and redirect output to client	see logfile in Appendix A.2.2	✓
get queue status message received	execute qstat locally and redirect output to client	see logfile in Appendix A.2.2	✓
start job message received	test if job start request is valid, if so allow first executor to start the job, suspend all executors for same job	see logfile in Appendix A.2.2	✓
job finished message received	release all pending executors	see logfile in Appendix A.2.2	✓
new JOSHUA server joins group	one of the group members dumps the current process state, new member replicates status, if something fails join is aborted else new member is joined	see logfile in Appendix A.2.2	✓
JOSHUA server fails	the failing server ends locally running PBS TORQUE and maui, remaining reconfigure new group, all running jobs stay uninterrupted, all user command still work with remaining group	see logfile in Appendix A.2.2	✓

Table 3.3.: System test JOSHUA server

capability	expected behavior	output verification (shortened) ...continuation	pass
external JOSHUA component fails	remaining external component processes get locally killed, JOSHUA server leaves group	see logfile in Appendix A.2.2	✓

Table 3.3.: System test JOSHUA server

capability	expected behavior	output verification (shortened)	pass
JOSHUA mutex	sends job started message, first executor is allowed to enter job (exit 0), remaining executors get suspended (exit 1 after job finished)	`Prologue Args: Job ID: 1.h1.mycluster.org User ID: kai Group ID: kai Starte jutex Exiting...0 done.`	✓
JOSHUA cluster job done	sends job finished message	`Epilogue Args: Job ID: 1.h2.mycluster.org User ID: kai Group ID: kai JOB name : test.sh`	✓

Table 3.4.: System test JOSHUA cluster mutex

4

Detailed Software Design

4.1 Job submission

The design of the symmetric Active/Active high availability solution for job-scheduler services provided by JOSHUA, though transparently applied has some impacts to the user.

Since the design of the *joshua* daemon takes advantage of the virtual paradigm and message oriented state replication processed trough an event-driven programming solution, there are limitation issues to the server application in terms of accessibility and scalability for the user commands.

The stress and performance tests already indicate that when under heavy load, a multi threaded server application possibly would handle the incoming requests better. But as dictated by the virtual synchrony paradigm a multi threaded solution is not compatible, since each message must be processed in the total incoming order, in order to preserve the states between each of the high available job scheduler nodes.

But impacts cannot only be observed, when *joshua* is under heavy under heavy load. As the performance tests showed, the HA capabilities have some cost as tradeoffs for the advanced high availability in comparison to a traditional single-headed PBS system.

Figure 4.1 illustrates in details how and where the JOSHUA job submission command differs from the normal PBS job submission.

Since the effective PBS command is executed on the head node itself, the input infor-

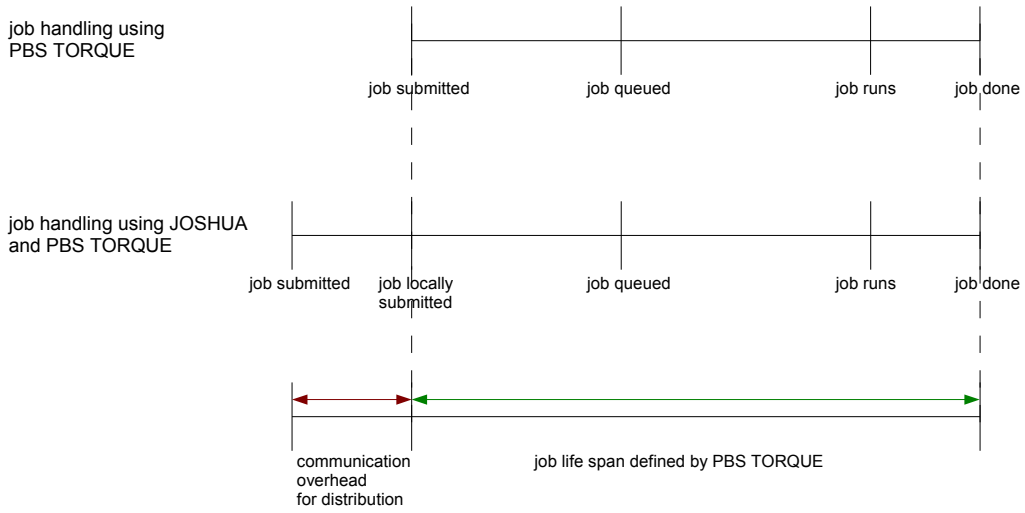

Figure 4.1.: User impacts on job submission

mation has to be transferred from the client to *joshua* daemon for further processing. The job submission is therefore prolonged because of the distribution of the client input data to all running daemons plus the time to redirect the command output from the server application back to the client.

4.2 Dynamic group reconfiguration

The members within the head-node group of *joshua* working in virtual synchrony must reconfigure their group dynamically in case of the join event and when a head node fails or leaves the group.

The failure model is rather simple, since all remaining members share the same process state and can therefore continue to operate without complex reconfiguration.

The remaining *joshua* daemons only need to update the member list array in order to know, who left and who remains within the group. This is even done internally by the transis group communication library. The member list is only used by *joshua* to ensure, that all members know, which process assists a new joining member in the join process.

In case of a join event, the algorithm splits into the task for one elected existing mem-

ber to dump its current state to the new joining process.

Since the communication between the existing and new process can also take advantage of the communication facilities of transis and its event-driven message processing, the algorithm for the join event can be rather simple as well.

The advantage of using the group communication facilities for the join data exchange is that no failure intolerant client/server socket interconnection has to be used. By using the message facilities, the already existing member dumps all its internal data to the joining process, which stays in an event-driven loop until the join process is finished.

The information exchange from the established member starts with a start join message followed by all join and delete messages stored internally by each process locally, followed by a finish join message.

The join and delete message are not a complete rollback of all messages ever received by the head-node group. The design of the internal data for *joshua* seeks as much memory saving as possible, since the server application is designed to run for a long time.

Only messages back to current queued jobs will be replicated to the joining member. That way the number of messages can be reduced to the necessary needed and the time spent in the join event for the existing member will be minimised as well. After all the messages are dumped via the group communication message system, the assisting process carries in with normal work. All messages will be stored by the transis daemon until the joining process fetches and process them.

The joining process loops within an event handler waiting for all necessary join messages before it will enter the final head-node group communication message handler. All pending messages for the head-node group will be delayed until the new process has processed all join messages to keep up to the current state of the assisting process, when it dumped its state.

After the join event is done, the new process will carry on and finally catch up with the rest of the virtual synchronous group to same level.

Before that the messages will be processed to gain the last process state of the assisting process by first setting all internal job data, followed by submitting all jobs to local queue, ending with deleting all deleted jobs to replicate also the holes in the job queue in order to eventually have the exact same local queue as every other member has.

After the last of these queues messages have been processed, the event handler for the join process will be left to enter the normal event handler to process the actual head-node group relevant messages with in the group. By catching up on these messages, the new process finally becomes a full member of the group.

If something goes wrong during the join process, the new joining process shuts down itself and all external components with it to ensure that no process member is online, which does not share the exact same global process state knowledge and local job queue in order to sustain the virtual synchrony.

4.3 Exchange of external components

As the design of JOSHUA intends every external runtime dependency can be replaced quite easy. As mentioned in the 3.2.1 every external component used by the JOSHUA software system is defined in a configuration file parsed by libconfuse.

This kind of dynamic configuration design makes it possible to replace the each external component when needed. If for example another job scheduler than Maui is needed on a certain site, only the configuration file has to be changed.

A replacement for Maui Cluster Scheduler^{TM}is rather easy to make, since JOSHUA does not directly interact with the job scheduler but indirect via the resource management system. In order to replace Maui with any other scheduler, the replacement has to be compatible with the resource manager but must support the in section 2.1.5 mentioned dedicated mode to not compromise the JOSHUA system design.

To replace the used resource management service system of PBS TORQUE more compatibility issues must be met in order to work properly. The replacement has to provide tools for job submission, deletion and status information. The queue manager also has to support a setup of multiple setups in order to enable the capabilities of the

proposed Active/Active high-availability architecture for services. Besides, in order to let *jmutex* operate as intended pre and post job scripts must be supported as well.

A possible replacement can be the Bamboo [Sca06] resource and queue manager as part of the Scalable Systems Software by Scalable Computing Laboratory [Ame06b] at Ames Laboratory [Ame06a].

So far Bamboo provides the basic features of a resource manager: [Sca06]

- PBS compatible syntax
- job submission
- job monitoring
- job signalling (send user signals to the users running jobs)
- job deletion
- job start-up (privileged command to start jobs)
- management of running jobs (administrative scripts, output delivery)

Since this is the first release of Bamboo, it has to be considered as an alpha release and thus there have to be expected bugs and incomplete features. [Sca06]

5 Conclusion

5.1 Results

This Master thesis project introduced a novel approach for high availability concerning the head node of a computational cluster. The proposed system design provides high availability to the job-scheduler service, as the most vital part of a cluster system setup.

The addition of high availability to the job-scheduler service and resource management system is achieved by using an existing group communication framework to manage virtual synchrony among multiple head nodes. The shared global state is, therefore, used to introduce a system providing Active/Active high availability leading to a significant increase of availability for cluster computing.

This dissertation proposed a new JOb Scheduler for High availability Using Active replication (JOSHUA). It takes advantage of existing software components commonly used in a cluster environment, such as the Portable Batch System. Thus, it transparently augments the job-scheduler service and resource management system with high availability.

All general system design tasks have been finished as final result of this Master thesis. As shown in the previous sections an overall system design to solve the key problems of the dissertation has been created.

A working environment has been build and setup for proper development and testing. The development was done on a small dedicated cluster with one to three head

nodes and a compute node. All nodes were homogeneous and identical in hardware and software setup.

To show how the proposed system design and its new form of symmetric Active/Active high availability can be accomplished in practice, a proof-of-concept implementation of JOSHUA has been developed. The results of this dissertation may be used for further improvement of high availability for cluster computation.

As part of the JOSHUA implementation, a utility library has been created, containing important capabilities for the JOSHUA components to work, including functions to handle signals and events, log, pack and unpack messages, and create linked lists for data storage.

The implemented solution for the problem of high availability for high-end scientific computing includes all components necessary to build a multi-head node cluster with symmetric Active/Active high availability. The components include a server daemon working on each head node, a bundle of user command line tools and a cluster node mutual exclusion application.

The capabilities of JOSHUA are limited to job submission, deletion and status information only, but proofs the intend proposition of the Active/Active high availability model design. The final solution is a first step toward non-stop computation by introducing a fault-tolerant job scheduler.

5.2 Future Work

The results and finding of this Master thesis are intended to enlighten the possibilities of higher forms of high availability even further.

A solution as proposed can possibly be used by the HA-OSCAR project to further improve their Active/Hot-Standby high availability solution.

There are several possibilities for improvements and research of the implemented design. To lift JOSHUA from a proof-of-concept application to a productive stable high availability solution the missing user features, such as holding and releasing

jobs must be added to the current design.

Some of the external components are not feasible enough to work in an industrial scale environment. The transis group communication framework has to be replaced by a more scalable and reliable communication subsystem such as openais [The05].

Since the used PBS TORQE version is just in the beginning of supporting multi-headed high availability environments, there are further improvements and demands to an external resource manager as well.

The benefits for a high-availability solution for high-end scientific computing would be greatly improved, when the so far only client/server model based Portable Batch System is more aware of the service level model used by the Active/Active high availability architecture for services.

References

[Alt03] Altair Grid technologies. Portable batch system. http://www.openpbs.org/, 2003. [Online; accessed 21-September-2005].

[Alt05] Altair Engineering, Inc. Altair pbs professional 7.0. http://www.altair.com/software/pbspro.htm, 2005. [Online; accessed 21-September-2005].

[Ame06a] Ames Laboratory. Ames laboratory homepage. http://www.ameslab.gov/, 2006. [Online; accessed 21-January-2006].

[Ame06b] Ames Laboratory. Scalable computing laboratory (scl). http://www.scl.ameslab.gov/, 2006. [Online; accessed 21-January-2006].

[Bar98] Jesús M. González Barahona. *A Communication Arquitecture for Process Groups*. PhD thesis, Universidad Politecnica de Madrid, February 1998. [Online; accessed 18-January-2006].

[Clu05a] Cluster Resources Inc. Torque resource manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php, 2005. [Online; accessed 21-September-2005].

[Clu05b] Cluster Resources Inc. Torque v2.0 admin manual. http://www.clusterresources.com/products/torque/docs20/torqueadmin.shtml, 2005. [Online; accessed 25-November-2005].

[Clu06] Cluster Resources Inc. Documentation maui cluster scheduler^{TM}. http://www.clusterresources.com/pages/resources/documentation.php, 2006. [Online; accessed 24-January-2006].

[Cor00] M. Corbatto. An introduction to portable batch system (pbs). http://hpc.sissa.it/pbs/pbs.html, 2000. [Online; accessed 06-November-2005].

[ES05] C. Engelmann and Stephen L. Scott. Concepts for high availability in scientific high-end computing. In *Proceedings of High Availability and Performance Computing Workshop (HAPCW)*, Santa Fe, NM, USA, October 2005. http://www.csm.ornl.gov/~engelman/publications/engelmann05concepts.pdf.

[ESI02] C. Engelmann, Stephen L. Scott, and G. A. Geist II. Distributed peer-to-peer control in harness. In *International Conference on Computational Science (2)*, pages 720–728, 2002. http://citeseer.ist.psu.edu/engelmann02distributed.html.

[ESLH06] Christian Engelmann, Stephen L. Scott, Chokchai Leangsuksun, and X. He. Active/active replication for highly available hpc system services. In *To appear in Proceedings of International Symposium on Frontiers in Availability, Reliability and Security (FARES)*, Vienna, Austria, April 2006.

[Eve05a] EventHelix.com Inc. Reliability and availability basics. http://www.eventhelix.com/RealtimeMantra/FaultHandling/reliability_availability_basics.htm, 2005. [Online; accessed 16-December-2005].

[Eve05b] EventHelix.com Inc. System reliability and availability. http://www.eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.htm, 2005. [Online; accessed 16-December-2005].

[Fre05] Free Software Foundation. Autoconf - gnu project - free software foundation (fsf). http://www.gnu.org/software/autoconf/, 2005. [Online; accessed 12-September-2005].

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, USA, 1994.

[HA-05] HA-OSCAR Group. High availability open source cluster application resources. http://xcr.cenit.latech.edu/ha-oscar/, 2005. [Online; accessed 29-September-2005].

[Hed05] Martin Hedenfalk. libconfuse. http://www.nongnu.org/confuse/, 2005. [Online; accessed 08-September-2005].

[Lin05] Linux-HA project. Heartbeat. http://linux-ha.org/HeartbeatProgram, 2005. [Online; accessed 24-January-2006].

[LML+05] C. Leangsuksun, V. K. Munganuru, T. Liu, S. L. Scott, and C. Engelmann. Asymmetric active-active high availability for high-end computing. In *Proceedings of 2nd International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2)*, Cambridge, MA, USA, June 2005. http://www.csm.ornl.gov/~engelman/publications/leangsuksun05asymmetric.pdf.

[LSL+03a] Box Leangsuksun, Lixin Shen, Tong Lui, Hertong Song, and Stephen L. Scott. Presentation at ieee cluster computing 2003. http://xcr.cenit.latech.edu/haoscar/cluster2003.pdf, December 2003. [Online; accessed 24-January-2006].

[LSL+03b] Chokchai Leangsuksun, Lixin Shen, Tong Liu, Herton Song, and Stephen L. Scott. Dependability prediction of high availability oscar cluster server. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '03*, Las Vegas, Nevada, USA, June 2003. http://xcr.cenit.latech.edu/ha-oscar/docs/PDPTA2003.pdf.

[Mal94] Dalia Malki. The transis user tutorial. ftp://ftp.cs.huji.ac.il/users/transis/tutorial.ps.gz, March 1994. [Online; accessed 01-October-2005].

[Mis05] Mission Critical Linux. Kimberlite. http://oss.missioncriticallinux.com/projects/kimberlite/, 2005. [Online; accessed 29-September-2005].

[MNS+03] John Muggler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, Jeffrey M. Squyres, Benoît des Ligneris, Francis Giraldeau, and Chokchai Leangsunksun. Oscar clusters. In *Proceedings of the Linux Symposium*, pages 387–397, Ottawa, Ontario, Canada, July 2003. http://citeseer.ist.psu.edu/711152.html.

[Ope05] Open Cluster Group. Open source cluster application resources. http://oscar.openclustergroup.org/, 2005. [Online; accessed 19-September-2005].

[Res96] Ron I. Resnick. A modern taxonomy of high availability. http:

//www.verber.com/mark/cs/systems/A%20Modern%20Taxonomy%20of%20High%20Availability.htm, 1996. [Online; accessed 14-September-2005].

[Rob05] Tim Robinson. High end scientific computing. http://www.ja.net/services/network-services/bmas/seminars/streaming_seminar/Intro_to_MC/img11.html, 2005. [Online; accessed 29-September-2005].

[Sca06] Scalable Systems Software Resource Management Working Group. Sss resource management and accounting: Bamboo. http://sss.scl.ameslab.gov/bamboo.shtml, 2006. [Online; accessed 21-January-2006].

[Ser05] Service Availability Forum. Service availabilty forum: home. http://www.saforum.org/home, 2005. [Online; accessed 21-September-2005].

[She02] Santosh Shetty. Determining the availability and reliability of storage configurations. http://www1.us.dell.com/content/topics/global.aspx/power/en/ps3q02_shetty?c=us&l=en&s=gen, August 2002. [Online; accessed 16-December-2005].

[Sil05] Silicon Graphics, Inc. Developer central open source - linux failsafe. http://oss.sgi.com/projects/failsafe/, 2005. [Online; accessed 29-September-2005].

[SOHL+96] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1996.

[Ste05] SteelEye Technology. Lifekeeper, high availability clustering, failover, data replication, disaster recovery, linux, windows. http://www.steeleye.com/, 2005. [Online; accessed 29-September-2005].

[The99a] The Ensemble distributed communication system. Ensemble is the next generation of the horus group communication toolkit. http://dsl.cs.technion.ac.il/projects/Ensemble/, 1999. [Online; accessed 01-October-2005].

[The99b] The Transis Project. Transis group communication system, that supports

efficient group multicast for high availability. http://www.cs.huji.ac.il/labs/transis/, 1999. [Online; accessed 01-October-2005].

[The04a] The HA-OSCAR Working Group. Ha-oscar cluster user manual. http://xcr.cenit.latech.edu/haoscar/HA_OSCAR_INSTALL_PA4.pdf, 2004. [Online; accessed 24-January-2006].

[The04b] Ed Thelen. Eniac. http://ed-thelen.org/comp-hist/ENIAC.html, 2004. [Online; accessed 06-January-2006].

[The05] The openais project. openais. http://developer.osdl.org/dev/openais/, 2005. [Online; accessed 21-September-2005].

[Val06] Valgrind Developers. Valgrind. http://valgrind.org/, 2006. [Online; accessed 21-September-2005].

[Ver00] Veridian Information Solutions, Inc. Portable batch system administrator guide. http://www.clusterresources.com/products/torque/docs/admin_guide.ps, 2000. [Online; accessed 06-November-2005].

[vH05] Dimitri van Heesch. Doxygen. www.doxygen.org/, 2005. [Online; accessed 12-September-2005].

[Wes05] Western Digital Corporation. Wd caviar re2 400 gb hard drives (wd4000yr). http://www.wdc.com/en/products/Products.asp?DriveID=158&Language=en, 2005. [Online; accessed 06-January-2006].

[Wik05a] Wikipedia. Computer cluster — wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Cluster_computing, 2005. [Online; accessed 29-September-2005].

[Wik05b] Wikipedia. Hochleistungsrechnen — wikipedia, the free encyclopedia. http://de.wikipedia.org/wiki/High_Performance_Computing, 2005. [Online; accessed 29-September-2005].

[Wik06a] Wikipedia. Load testing — wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Load_testing, 2006. [Online; accessed 31-January-2006].

[Wik06b] Wikipedia. Software testing — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Software_testing`, 2006. [Online; accessed 31-January-2006].

[Wik06c] Wikipedia. System testing — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/System_testing`, 2006. [Online; accessed 31-January-2006].

A
Appendix

A.1 Manual

A.1.1 Installation

Since the JOSHUA software system takes advantage of the Autotools [Fre05] of the Free Software Foundation, Inc., the installation follows their basic instructions.

Basic Installation

These are generic installation instructions.

The ‘configure’ shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a ‘Makefile’ in each directory of the package. It may also create one or more ‘.h’ files containing system-dependent definitions. Finally, it creates a shell script ‘config.status’ that you can run in the future to recreate the current configuration, and a file ‘config.log’ containing compiler output (useful mainly for debugging ‘configure’).

It can also use an optional file (typically called ‘config.cache’ and enabled with ‘–cache-file=config.cache’ or simply ‘-C’) that saves the results of its tests to speed up reconfiguring. (Caching is disabled by default to prevent problems with accidental use of stale cache files.)

If you need to do unusual things to compile the package, please try to figure out how ‘configure’ could check whether to do them, and mail diffs or instructions to the

address given in the 'README' so they can be considered for the next release. If you are using the cache, and at some point 'config.cache' contains results you don't want to keep, you may remove or edit it.

The file 'configure.ac' (or 'configure.in') is used to create 'configure' by a program called 'autoconf'. You only need 'configure.ac' if you want to change it or regenerate 'configure' using a newer version of 'autoconf'.

The simplest way to compile this package is:

1. 'cd' to the directory containing the package's source code and type './configure' to configure the package for your system. If you're using 'csh' on an old version of System V, you might need to type 'sh ./configure' instead to prevent 'csh' from trying to execute 'configure' itself. Running 'configure' takes awhile. While running, it prints some messages telling which features it is checking for.

2. Type 'make' to compile the package.

3. Optionally, type 'make check' to run any self-tests that come with the package.

4. Type 'make install' to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that 'configure' created (so you can compile the package for a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

Prerequisites

In order to compile and install the JOSHUA software package the following libraries and header files must be installed:

- *libconfuse*

- *libtransis*

The configure script will automatically search for the necessary libraries before you can compile the sources. If the the configuration script is unable to resolve the dependencies it will print an error message.

Installation instructions

Following the basic installation guidelines you have to apply the following commands to compile the source code and install the binaries:

- cd /tmp
- tar -xzf joshua-01.tar.gz
- cd joshua-0.1
- ./configure
- make all
- make install

A.1.2 Usage

In order to use the JOSHUA software system the following runtime components must be setup and installed on your system:

- JOSHUA server application (*joshua*)
- job scheduler (for example Maui Cluster Scheduler)
- resource management server daemon (e.g. PSB TORQUE server)
- transis group communication daemon
- *libconfuse* runtime library

Especially the resourc management executor (e.g. PBS mom) must be equipped with the pro- and epiloque scripts for the proper execution of the cluster mutex. See A.3.2 and A.3.2 for examples.

For the proper setup of the external components see their installation and configuration manuals.

All components of JOSHUA use a configuration file. See example in A.1.3 for details.

Generally, JOSHUA augments the external resource management and job-scheduler service with high availability. With some restrictions concerning the user tool options, the system usage is quite like the used external components for your chosen cluster setup.

Server application

To start the server application the transis group communication daemon must be up and running. Simply start the server application and all its external components by typing 'jinit -c configfile'.

If everything works fine, jinit will return the following out put ti the standard output:

```
linux# ./jinit -c /etc/joshua/joshua.conf
jinit started...
Attemting to start JOSHUA components...
job scheduler.............. done
joshua..................... done
resource manager.......... done
jobserver.................. done
```

Additional information can always be obtained during runtime trough the error- and logfiles specified in the configuration file.

User commands

The user commands need a running joshua server daemon to work. All head nodes available on your site can be determined in the configuration file (see A.1.3 for details).

Examples for user command usage:

Capability	Command	Output
get queue status and redirect output from PBS	*jstat*	`Job id Name User Time Use S Queue ---------- -------- -------- ---- - --- 1.h1 test.sh kai 0 R batch`
submit job with jobscript and return job identifier	*jsub script.sh*	`0.h1.mycluster.org`
submit job via STDIN and return job identifier	*echo "hostname" | jsub*	`1.h1.mycluster.org`
delete job by job identifier	*jdel 1*	no output

Table A.1.: User command examples

A.1.3 JOSHUA configuration file example

```
1 headnodes = {"joshua.mycluster.org", "h1.mycluster.org", "h2.mycluster.org"}
  logfile = "/tmp/joshua.log"
3 errorlog = "/tmp/joshuaerr.log"
  scheduler_exec = "/usr/local/maui/sbin/maui"
5 scheduler_conf = "/usr/local/maui/maui.cfg"
  scheduler = "maui"
7 job_server_exec = "/usr/local/sbin/pbs_server"
  job_server_conf = "/var/spool/server_priv/serverdb"
9 job_server = "pbs_server"
  group_com_exec = "/usr/local/sbin/transis"
11 group_com_conf = "/etc/transis/transis.conf"
  group_com = "transis"
13 joshua_exec = "/home/kai/joshua-0.1/joshua/joshua"
  joshua_conf = "/etc/joshua/joshua.conf"
15 joshua = "joshua"
  submit_exec = "/usr/local/bin/qsub"
17 del_exec = "/usr/local/bin/qdel"
```

```
stat_exec = "/usr/local/bin/qstat"
```

A.2 Test output

A.2.1 Memory allocation test output

```
==16023== Memcheck, a memory error detector for x86-linux.
==16023== Copyright (C) 2002-2005, and GNU GPL'd, by Julian Seward et al.
==16023== Using valgrind-2.4.0, a program supervision framework for x86-linux.
==16023== Copyright (C) 2000-2005, and GNU GPL'd, by Julian Seward et al.
==16023==
==16023== My PID = 16023, parent PID = 15909.  Prog and args are:
==16023==    ./joshua
==16023==    -c
==16023==    /etc/joshua/joshua.conf
==16023==
==16023== Valgrind library directory: /usr/lib/valgrind
==16023== Command line
==16023==    ./joshua
==16023==    -c
==16023==    /etc/joshua/joshua.conf
==16023== Startup, with flags:
==16023==    --leak-check=yes
==16023==    --show-reachable=yes
==16023==    -v
==16023==    --log-file=/tmp/joshua
==16023== Contents of /proc/version:
==16023==   Linux version 2.6.8-2-686-smp (horms@tabatha.lab.ultramonkey.org) (gcc version 3.
    3.5 (Debian 1:3.3.5-12)) #1 SMP Thu May 19 17:27:55 JST 2005
==16023== Reading syms from /home/kai/joshua-0.1/joshua/joshua (0x8048000)
==16023== Reading syms from /lib/ld-2.3.2.so (0x1B8E4000)
==16023==    object doesn't have a symbol table
==16023== Reading debug info from /lib/ld-2.3.2.so...
==16023== ... CRC mismatch (computed E7117123 wanted 4ECF6D33)
==16023==    object doesn't have any debug info
==16023== Reading syms from /usr/lib/valgrind/stage2 (0xB0000000)
==16023== Reading syms from /lib/ld-2.3.2.so (0xB1000000)
==16023==    object doesn't have a symbol table
==16023== Reading debug info from /lib/ld-2.3.2.so...
==16023== ... CRC mismatch (computed E7117123 wanted 4ECF6D33)
==16023==    object doesn't have any debug info
==16023== Reading syms from /lib/tls/libdl-2.3.2.so (0xB101D000)
==16023==    object doesn't have a symbol table
==16023== Reading debug info from /lib/tls/libdl-2.3.2.so...
==16023== ... CRC mismatch (computed 71527790 wanted 2DA21AD9)
==16023==    object doesn't have any debug info
==16023== Reading syms from /lib/tls/libc-2.3.2.so (0xB1020000)
==16023==    object doesn't have a symbol table
==16023== Reading debug info from /lib/tls/libc-2.3.2.so...
==16023== ... CRC mismatch (computed 9FEA8425 wanted 656F7E39)
==16023==    object doesn't have any debug info
==16023== Reading syms from /usr/lib/valgrind/vgskin_memcheck.so (0xB1256000)
==16023== Reading suppressions file: /usr/lib/valgrind/default.supp
==16023==
==16023== Reading syms from /usr/lib/valgrind/vg_inject.so (0x1B8FE000)
==16023== Reading syms from /usr/lib/valgrind/vgpreload_memcheck.so (0x1B901000)
==16023== Reading syms from /lib/tls/libm-2.3.2.so (0x1B911000)
==16023==    object doesn't have a symbol table
```

```
52 ==16023== Reading debug info from /lib/tls/libm-2.3.2.so...
   ==16023== ... CRC mismatch (computed F11D9E14 wanted 49928816)
54 ==16023==    object doesn't have any debug info
   ==16023== Reading syms from /usr/lib/libconfuse.so.0.0.0 (0x1B934000)
56 ==16023==    object doesn't have a symbol table
   ==16023==    object doesn't have any debug info
58 ==16023== Reading syms from /lib/tls/libc-2.3.2.so (0x1B93F000)
   ==16023==    object doesn't have a symbol table
60 ==16023== Reading debug info from /lib/tls/libc-2.3.2.so...
   ==16023== ... CRC mismatch (computed 9FEA8425 wanted 656F7E39)
62 ==16023==    object doesn't have any debug info
   ==16023== Reading syms from /lib/tls/libdl-2.3.2.so (0x1BA75000)
64 ==16023==    object doesn't have a symbol table
   ==16023== Reading debug info from /lib/tls/libdl-2.3.2.so...
66 ==16023== ... CRC mismatch (computed 71527790 wanted 2DA21AD9)
   ==16023==    object doesn't have any debug info
68 ==16023== TRANSLATE: 0x1B9B0BB0 redirected to 0x1B904510
   ==16023== TRANSLATE: 0x1B9B0E00 redirected to 0x1B904FA1
70 ==16023== TRANSLATE: 0x1B9B0D40 redirected to 0x1B904A82
   ==16023== TRANSLATE: 0x1B9B8CD0 redirected to 0x1B905C80
72 ==16023== TRANSLATE: 0x1B9B73E0 redirected to 0x1B9057B0
   ==16023== TRANSLATE: 0x1B9B12C0 redirected to 0x1B904EE2
74 ==16023==
   ==16023== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 17 from 1)
76 --16023--
   --16023-- supp:   17 Ugly strchr error in /lib/ld-2.3.2.so
78 ==16023== malloc/free: in use at exit: 17352 bytes in 8 blocks.
   ==16023== malloc/free: 3529 allocs, 3521 frees, 344831 bytes allocated.
80 ==16023==
   ==16023== searching for pointers to 8 not-freed blocks.
82 ==16023== checked 328644 bytes.
   ==16023==
84 ==16023== 16 bytes in 2 blocks are still reachable in loss record 1 of 5
   ==16023==    at 0x1B90459D: malloc (vg_replace_malloc.c:130)
86 ==16023==    by 0x804E325: zzz_Connect (zzz_layer.c:182)
   ==16023==    by 0x8049658: main (joshua.c:108)
88 ==16023==
   ==16023==
90 ==16023== 30 bytes in 2 blocks are still reachable in loss record 2 of 5
   ==16023==    at 0x1B90459D: malloc (vg_replace_malloc.c:130)
92 ==16023==    by 0x804D223: HA_Connect (ha.c:186)
   ==16023==    by 0x804E0AF: ha_connect (zzz_layer.c:68)
94 ==16023==    by 0x804E384: zzz_Connect (zzz_layer.c:190)
   ==16023==    by 0x8049658: main (joshua.c:108)
96 ==16023==
   ==16023==
98 ==16023== 82 bytes in 1 blocks are still reachable in loss record 3 of 5
   ==16023==    at 0x1B90459D: malloc (vg_replace_malloc.c:130)
100 ==16023==   by 0x8049C89: handle_events (server.c:133)
   ==16023==    by 0x804C53B: do_user_fds (events.c:275)
102 ==16023==   by 0x804CC2D: E_dispatch_events (events.c:575)
   ==16023==    by 0x804CD23: E_main_loop (events.c:633)
104 ==16023==   by 0x8049729: main (joshua.c:153)
   ==16023==
106 ==16023==
   ==16023== 1000 bytes in 1 blocks are still reachable in loss record 4 of 5
108 ==16023==   at 0x1B904F75: calloc (vg_replace_malloc.c:175)
   ==16023==    by 0x804C1D6: new_sq_node (events.c:84)
110 ==16023==   by 0x804C7FE: E_add_sock (events.c:407)
   ==16023==    by 0x804DE5C: HA_Add_Upcall (ha.c:586)
112 ==16023==   by 0x804970F: main (joshua.c:148)
   ==16023==
114 ==16023==
```

```
==16023== 16224 bytes in 2 blocks are still reachable in loss record 5 of 5
==16023==    at 0x1B90459D: malloc (vg_replace_malloc.c:130)
==16023==    by 0x804CF06: HA_Connect (ha.c:126)
==16023==    by 0x804E0AF: ha_connect (zzz_layer.c:68)
==16023==    by 0x804E384: zzz_Connect (zzz_layer.c:190)
==16023==    by 0x8049658: main (joshua.c:108)
==16023==
==16023== LEAK SUMMARY:
==16023==    definitely lost: 0 bytes in 0 blocks.
==16023==      possibly lost: 0 bytes in 0 blocks.
==16023==    still reachable: 17352 bytes in 8 blocks.
==16023==         suppressed: 0 bytes in 0 blocks.
--16023--     TT/TC: 0 tc sectors discarded.
--16023--            6536 tt_fast misses.
--16023-- translate: new     4894 (80866 -> 1077502; ratio 133:10)
--16023--            discard 0 (0 -> 0; ratio 0:10).
--16023-- chainings: 3643 chainings, 0 unchainings.
--16023--  dispatch: 15619093 jumps (bb entries); of them 675504 (4%) unchained.
--16023--            313/20402 major/minor sched events.
--16023-- reg-alloc: 915 t-req-spill, 191914+6486 orig+spill uis,
--16023--            25423 total-reg-rank
--16023--    sanity: 314 cheap, 13 expensive checks.
--16023--    ccalls: 18276 C calls, 55% saves+restores avoided (59886 bytes)
--16023--            24918 args, avg 0.87 setup instrs each (6416 bytes)
--16023--            0% clear the stack (54579 bytes)
--16023--            7517 retvals, 30% of reg-reg movs avoided (4442 bytes)
```

A.2.2 System tests

JOSHUA startup logfile excerpt

```
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29375= Info: Inititiate Server startup
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29375= Info: Startup finished.
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: jbootup started...
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Waiting till joshua is ready..
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29378= Info: joshua started
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Inititiate Server startup
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Startup finished.
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: I'm there
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: JOSHUA daemon started...
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size = 46
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size = 40
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++ Group change in group headmasters from 1 to 1 member(s)
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: headmaster_joshua is the first among the headnodes
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Attempting to open /var/spool/server_priv/serverdb
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Send signal, that structure is set
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29379= Info: pbs_server started
ping_nodes: entered
ping_nodes: ping h3
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Sending signal that pbs is running
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Signal send
```

```
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29381= Info: jobserver started
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: Waiting for bootup.. to contiue
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: Got notification lets go on
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: done.
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: headmaster_joshua is now master
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: + Members so far:
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: + headmaster_joshua
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
```

JOSHUA submission event logfile excerpt

```
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size =
    993
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: DEBUG: /home/kai/joshua-0.1/jcmd/jsub
[Thu Feb  9 16:06:23 2006] joshua =29395= Info: Created child with pid 29395 to exec /usr/
    local/bin/qsub command
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: exec returned stdout
0.joshua.ornl.gov

[Thu Feb  9 16:06:23 2006] joshua =29378= Info: exec returned stderr
(null)
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: Return to sender 113951918285733_joshua
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: Added job 0 to internal submission queue.
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: Sending..
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: done..
[Thu Feb  9 16:06:23 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
```

JOSHUA deletion event logfile excerpt

```
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: ++++ Event ++++ Message received ++++ size =
    992
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: DEBUG: ./jdel
[Tue Feb 14 12:38:53 2006] joshua =4805= Info: Created child with pid 4805 to exec /usr/local
    /bin/qdel command
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: exec returned stdout
(null)
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: exec returned stderr
(null)
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: Return to sender 113993873347367_joshua
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: Added job 3 to internal deletion queue.
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: Sending..
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: done..
[Tue Feb 14 12:38:53 2006] joshua =4760= Info: ++++ Event ++++ handled ++++
```

JOSHUA status event logfile excerpt

```
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: ++++ Event ++++ Message received ++++ size =
    989
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: DEBUG: ./jstat
[Tue Feb 14 12:38:57 2006] joshua =4808= Info: Created child with pid 4808 to exec /usr/local
    /bin/qstat command
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: exec returned stdout
Job id                    Name             User            Time Use S Queue
------------------------  ---------------- --------------- -------- - -----
1.joshua                  test.sh          kai             00:00:00 R batch
2.joshua                  test.sh          kai                    0 Q batch
4.joshua                  test.sh          kai                    0 Q batch
```

```
5.joshua              test.sh          kai                    0 Q batch

[Tue Feb 14 12:38:57 2006] joshua =4760= Info: exec returned stderr
(null)
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: Return to sender 113993873762460_joshua
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: Sending..
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: done..
[Tue Feb 14 12:38:57 2006] joshua =4760= Info: ++++ Event ++++ handled ++++
```

JOSHUA job start event logfile excerpt

```
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: ++++ Event ++++ Message received ++++ size =
    39
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: Received start message for job 1 from
    113993877893447_joshua
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: LDONE: -1 JID: 1 LSUB: 1 GENDONE: 0 GENSUB: 0
    --> Sanity: OK
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: LDONE: -1 JID: 1 LSUB: 1 GENDONE: 0 GENSUB: 0
    --> Exec: OK
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: Executor 113993877893447_joshua has allowance
    to enter job 1
[Tue Feb 14 12:38:43 2006] joshua =4760= Info: ++++ Event ++++ handled ++++
```

JOSHUA job finished event logfile excerpt

```
[Tue Feb 14 12:44:10 2006] joshua =4827= Info: ++++ Event ++++ Message received ++++ size =
    29
[Tue Feb 14 12:44:10 2006] joshua =4827= Info: Received finish message for job 1 from jdone_
    joshua
[Tue Feb 14 12:44:10 2006] joshua =4827= Info: ++++ Event ++++ handled ++++
```

JOSHUA join event logfile excerpt

```
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29375= Info: Inititiate Server
    startup
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29375= Info: Startup finished.
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: jbootup started...
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Waiting till
    joshua is ready..
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29378= Info: joshua started
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Inititiate Server startup
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Startup finished.
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: I'm there
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: JOSHUA daemon started...
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size =
    46
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size =
    40
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: ++ Group change in group headmasters from 1
    to 1 member(s)
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: headmaster_joshua is the first among the
    headnodes
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Attempting to open /var/spool/server_priv/
    serverdb
[Thu Feb  9 16:05:44 2006] joshua =29378= Info: Send signal, that structure is set
[Thu Feb  9 16:05:44 2006] /home/kai/joshua-0.1/joshua/jinit =29379= Info: pbs_server started
ping_nodes: entered
ping_nodes: ping h3
```

```
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Sending signal that pbs is running
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29376= Info: Signal send
[Thu Feb  9 16:05:45 2006] /home/kai/joshua-0.1/joshua/jinit =29381= Info: jobserver started
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: Waiting for bootup.. to contiue
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: Got notification lets go on
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: done.
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: headmaster_joshua is now master
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: + Members so far:
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: + headmaster_joshua
[Thu Feb  9 16:05:45 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size = 57
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: ++ Group change in group headmasters from 1 to 2 member(s)
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: Assisting the new member the joining process
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: Send FINISH JOIN
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: all join data sent..
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: headmaster_h1 is now master
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: + Members so far:
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: + headmaster_h1
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: + headmaster_joshua
[Thu Feb  9 16:05:56 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size = 74
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: ++ Group change in group headmasters from 2 to 3 member(s)
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: I am not of any help headmaster_joshua headmaster_h1
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: headmaster_h1 is now master
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: + Members so far:
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: + headmaster_h1
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: + headmaster_h2
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: + headmaster_joshua
[Thu Feb  9 16:06:02 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: ++++ Event ++++ Message received ++++ size = 91
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: ++ Group change in group headmasters from 3 to 4 member(s)
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: I am not of any help headmaster_joshua headmaster_h1
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: headmaster_h1 is now master
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: + Members so far:
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: + headmaster_h1
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: + headmaster_h2
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: + headmaster_h3
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: + headmaster_joshua
[Thu Feb  9 16:06:07 2006] joshua =29378= Info: ++++ Event ++++ handled ++++
```

JOSHUA failure logfile excerpt

```
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: ++++ Event ++++ Message received ++++ size = 40
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: ++ Group change in group headmasters from 2 to 1 member(s)
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: headmaster_joshua is now master
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: + Members so far:
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: + headmaster_joshua
[Tue Feb 14 12:44:48 2006] joshua =4827= Info: ++++ Event ++++ handled ++++
```

JOSHUA extern failure event logfile excerpt

```
[Tue Feb 14 12:39:13 2006] ./jinit =4758= Warning: Child exited. Shutdown initiated.
[Tue Feb 14 12:39:13 2006] ./jinit =4762= Warning: Broken pipe. Shutdown initiated.
```

A.3 Sources code listings

A.3.1 jcmd

jsub.c

```
/*****************************************************************************
* Project: JOSHUA                                                            *
* Description: Command line tool to jsub to submit/add jobs similar qsub     *
* Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                              *
*                                                                            *
*           888888                    888                                    *
*             "88b                    888                                    *
*              888                    888                                    *
*              888  .d88b.  .d8888b  88888b.  888  888  8888b.               *
*              888 d88""88b 88K      888 "88b 888  888     "88b              *
*              888 888  888 "Y8888b. 888  888 888  888 .d888888              *
*              88P Y88..88P      X88 888  888 Y88b 888 888  888              *
*              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888              *
*            .d88P                                                           *
*          .d88P"                              2006 Kai Uhlemann             *
*         888P"                                                              *
*                                                                            *
* Created at: Mon Nov  7 10:58:14 EST 2005                                   *
* System: Linux 2.6.8-2-686-smp on i686                                      *
*                                                                            *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
*                                                                            *
*****************************************************************************/
/*****************************************************************************
*                                                                            *
* Headers                                                                    *
*                                                                            *
*****************************************************************************/
#include "utils.h"
/*****************************************************************************
*                                                                            *
* Global data                                                                *
*                                                                            *
*****************************************************************************/
static zzz_mbox_cap      msgbox;
/*****************************************************************************
*                                                                            *
* Prototypes                                                                 *
*                                                                            *
*****************************************************************************/
void handle_events(int, void *);
void timeout(void*);
/*****************************************************************************
*                                                                            *
* Main                                                                       *
```

```
46 *                                                                            *
   ******************************************************************************/
48 int main(int argc, char **argv, char **env)
   {
50 /*
   * Local data
52 */
     /* Transis data */
54   int flag = 1;
     char *stack = NULL; /* default layer stack will be used */
56   static mbox_cap     ha_msgbox;

58   char *sender=NULL;
     char *msg=NULL;
60   char *input=NULL;
     char *path=NULL;
62   char **serverlist;
     int ret=0;
64   struct timeval tout, t_current;
     int entries=0, i=0;
66
     /* used to create username */
68   double current;
     char* coname=NULL;
70
   #ifdef SYSCONFDIR
72   char *conf = EXPAND(SYSCONFDIR);
   #else
74   char *conf = NULL;
     fprintf(stderr, "Error: No default sysconfdir defined\n");
76   exit(EXIT_FAILURE);
   #endif
78
     /* check for sane input, accept no options so far */
80   if(argc>2)
     {
82     fprintf(stderr,"Error: Option not supported.\n");
       exit(EXIT_FAILURE);
84   }

86   /* options for the config file */
         cfg_opt_t opts[] =
88   {

90     CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
       CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
92     CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
       CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
94     CFG_STR("scheduler", "maui", CFGF_NONE),
       CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
96     CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
       CFG_STR("job_server", "pbs_server", CFGF_NONE),
98     CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
       CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
100    CFG_STR("group_com", "transis", CFGF_NONE),
       CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
102    CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
       CFG_STR("joshua", "joshua", CFGF_NONE),
104    CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
       CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
106    CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
       /* a memory leak in libconfuse forces to leave the default to NULL */
108    CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
```

```
    CFG_END()
  };
  cfg_t *cfg;

  cfg = cfg_init(opts, CFGF_NONE);

  if(cfg_parse(cfg, conf) == CFG_PARSE_ERROR)
  {
    /* no output will be seen when uninitialized */
    fprintf(stderr, "Error: parsing configfile\n");
    exit(EXIT_FAILURE);
  }

  /* alloc server list */
  entries=cfg_size(cfg, "headnodes");
  if(entries<1)
  {
    fprintf(stderr, "No headnode entries in config file %s\n", conf);
    cfg_free(cfg);
    exit(EXIT_FAILURE);
  }

  serverlist=(char **)malloc(sizeof(char*)*(entries+1));
  if(serverlist==NULL)
  {
    fprintf(stderr, "malloc %s\n",strerror(errno));
    exit(EXIT_FAILURE);
  }

  /* servers */
  for(i=0; i<entries; i++)
  {
    serverlist[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
  }
  serverlist[i]=NULL;


  /* free config file structure */
  cfg_free(cfg);


  /* set the timeout values */
  tout.tv_sec=10;
  tout.tv_usec=0;

  /* create connection name from clockticks since 1970 */
  gettimeofday(&t_current, NULL);
  current=t_current.tv_sec*1000000.0+t_current.tv_usec;

  /* get the string size for the clock ticks and allocate space */
  if((coname=(char*)malloc((snprintf(NULL, 0, "%.0lf",
current)+1)*sizeof(char)))==NULL)
  {
    log_err("malloc %s\n", strerror(errno));
  }

  /* create the connection name */
  snprintf(coname, snprintf(NULL, 0, "%.0lf", current)+1, "%.0lf",
current);

  /* get the cwd */
  path=gcwd();
```

```
172   /* get the stdin */
      if(chkstdin()==0)
174   {
        input=readfd(STDIN_FILENO);
176   }

178   /* connect remotely to transis */
      for(i=0; i<entries; i++)
180   {
        if((msgbox = zzz_RemoteConnect( coname, stack, flag, serverlist[i],
182 4001))!=0)
        {
184       break;
        }
186   }
      /* if none of the listed head nodes was reachable exit */
188   if(msgbox==0)
      {
190     for(i=0; i<entries; i++)
        {
192       fprintf(stderr,"Error: RemoteConnect to host %s connection failed.\n",
    serverlist[i]);
194     }
        destroylist(&serverlist);
196     exit(EXIT_FAILURE);
      }
198   /* we dont need that string anymore */
      free(coname);
200
      /* free server list */
202   destroylist(&serverlist);

204   /* change focus */
      ha_msgbox = zzz_Focus (msgbox, "HA");
206
      /* get sender name */
208   sender= HA_Get_Logical_Name(ha_msgbox);

210   E_init();

212   /* put together a message includeing all execution details */
      msg=mkaddmsg(getuid(), getgid(), input, argv, env, sender, path);
214   if(strlen(msg)>MAX_MSG_SIZE)
      {
216     fprintf(stderr, "Message to remote server too long. Try less STDIN.\n");
        exit(EXIT_FAILURE);
218   }

220   destroystring(&input);
      destroystring(&path);
222
      ret = zzz_VaSend(msgbox, SAFE, 0, strlen(msg)+1 , msg,
224 HEADNODEGROUP, NULL);

226   if(ret<strlen(msg))
      {
228     fprintf(stderr, "Message sent failed.\n");
        exit(EXIT_FAILURE);
230   }

232   destroystring(&msg);

234
```

```
  /* add event base for incoming messages */
236 zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
  E_sched(tout, timeout, (void *) 1);
238
  /* start event handler */
240 E_main_loop();

242 return(EXIT_SUCCESS);

244 }

246
/*****************************************************************************
248 *                                                                           *
  * handle_events                                                             *
250 *                                                                           *
  *****************************************************************************/
252 void handle_events(int dummy1, void *param)
  {
254 /*
  * local data
256 */
  /* message buffer for TRANSIS msgs */
258 char recv_buf[MAX_MSG_SIZE];
  int recv_type, amount;
260 char *out=NULL, *err=NULL;
  view *gview;
262

264 amount=zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
  /* distinguish between group chang msg and data msg */
266 /* data msg, dont care for group msgs */
  if( recv_type != VIEW_CHANGE)
268 {
    /* first recover the msgid */
270 switch(recov_id(recv_buf))
    {
272   /* message received was an submit msg*/
      case(RSPMSGID):
274     err=recov_stderr(recv_buf);
  out=recov_stdout(recv_buf);
276     if(out!=NULL)
        {
278       fprintf(stdout, "%s", out);
        }
280     else
  {
282   if(err!=NULL)
        {
284       fprintf(stderr, "%s", err);
        }
286 }
      destroystring(&out);
288   destroystring(&err);
      exit(EXIT_SUCCESS);
290   break;
    /* message unknown and is being ignored */
292 case(UNKNOWNMSG):
      fprintf(stderr, "Unknown message: %s\n", recv_buf);
294   exit(EXIT_FAILURE);
  }
296 }
```

```
298 }

300 void timeout(void *param)
{
302     fprintf(stderr, "Operation timed out...\n");
    exit(EXIT_FAILURE);
304
}
```

jdel.c

```
/*****************************************************************************
2 * Project: JOSHUA                                                          *
* Description: Command line tool jdel to delete jobs, similar qsub          *
4 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                           *
*                                                                           *
6 *           888888                    888                                 *
*             "88b                      888                                 *
8 *              888                    888                                 *
*              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
10 *             888 d88""88b 88K      888 "88b 888  888     "88b            *
*              888 888  888 "Y8888b. 888  888 888  888 .d888888            *
12 *             88P Y88..88P      X88 888  888 Y88b 888 888  888            *
*              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888            *
14 *            .d88P                                                       *
*           .d88P"                           2006 Kai Uhlemann            *
16 *         888P"                                                         *
*                                                                           *
18 * Created at: Mon Nov  7 10:58:14 EST 2005                                *
* System: Linux 2.6.8-2-686-smp on i686                                     *
20 *                                                                         *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
22 *                                                                         *
*****************************************************************************/
24 /*****************************************************************************
*                                                                           *
26 * Headers                                                                 *
*                                                                           *
28 *****************************************************************************/
#include "utils.h"
30 /*****************************************************************************
*                                                                           *
32 * Global data                                                             *
*                                                                           *
34 *****************************************************************************/
static zzz_mbox_cap     msgbox;
36 /*****************************************************************************
*                                                                           *
38 * Prototypes                                                              *
*                                                                           *
40 *****************************************************************************/
void handle_events(int, void *);
42 void timeout(void*);
/*****************************************************************************
44 *                                                                           *
* Main                                                                      *
46 *                                                                           *
*****************************************************************************/
48 int main(int argc, char **argv, char **env)
{
50 /*
* Local data
```

```
52 */
   /* Transis data */
54 int flag = 1;
   char *stack = NULL; /* default layer stack will be used */
56 static mbox_cap     ha_msgbox;

58 char *sender=NULL;
   char *msg=NULL;
60 char *input=NULL;
   char *path=NULL;
62 char **serverlist;
   int ret=0;
64 struct timeval tout, t_current;
   int entries=0, i=0;
66
   /* used to create username */
68 double current;
   char* coname=NULL;
70
#ifdef SYSCONFDIR
72 char *conf = EXPAND(SYSCONFDIR);
#else
74 char *conf = NULL;
   fprintf(stderr, "Error: No default sysconfdir defined\n");
76 exit(EXIT_FAILURE);
#endif
78
   /* check for sane input, accept no options so far */
80 if(argc>2)
   {
82     fprintf(stderr,"Error: Multiple job deletion not supported or option not\
supported.\n");
84     exit(EXIT_FAILURE);
   }
86
   /* options for the config file */
88     cfg_opt_t opts[] =
   {
90
     CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
92   CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
     CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
94   CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
     CFG_STR("scheduler", "maui", CFGF_NONE),
96   CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
     CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
98   CFG_STR("job_server", "pbs_server", CFGF_NONE),
     CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
100  CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
     CFG_STR("group_com", "transis", CFGF_NONE),
102  CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
     CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
104  CFG_STR("joshua", "joshua", CFGF_NONE),
     CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
106  CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
     CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
108   /* a memory leak in libconfuse forces to leave the default to NULL */
     CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
110  CFG_END()
   };
112 cfg_t *cfg;

114 cfg = cfg_init(opts, CFGF_NONE);
```

```
116   if(cfg_parse(cfg, conf) == CFG_PARSE_ERROR)
      {
118     /* no output will be seen when uninitialized */
        fprintf(stderr, "Error: parsing configfile\n");
120     exit(EXIT_FAILURE);
      }
122
      /* alloc server list */
124   entries=cfg_size(cfg, "headnodes");
      if(entries<1)
126   {
        fprintf(stderr, "No headnode entries in config file %s\n", conf);
128     cfg_free(cfg);
        exit(EXIT_FAILURE);
130   }

132   serverlist=(char **)malloc(sizeof(char*)*(entries+1));
      if(serverlist==NULL)
134   {
        fprintf(stderr, "malloc %s\n",strerror(errno));
136     exit(EXIT_FAILURE);
      }
138
      /* servers */
140   for(i=0; i<entries; i++)
      {
142     serverlist[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
      }
144   serverlist[i]=NULL;

146
      /* free config file structure */
148   cfg_free(cfg);

150
      /* set the timeout values */
152   tout.tv_sec=10;
      tout.tv_usec=0;
154
      /* create connection name from clockticks since 1970 */
156   gettimeofday(&t_current, NULL);
      current=t_current.tv_sec*1000000.0+t_current.tv_usec;
158
      /* get the string size for the clock ticks and allocate space */
160   if((coname=(char*)malloc((snprintf(NULL, 0, "%.01f",
    current)+1)*sizeof(char)))==NULL)
162   {
        log_err("malloc %s\n", strerror(errno));
164   }

166   /* create the connection name */
      snprintf(coname, snprintf(NULL, 0, "%.01f", current)+1, "%.01f",
168 current);

170   /* get the cwd */
      path=gcwd();
172
      /* get the stdin */
174   if(chkstdin()==0)
      {
176     input=readfd(STDIN_FILENO);
      }
```

```
178
    /* connect remotely to transis */
180 for(i=0; i<entries; i++)
    {
182     if((msgbox = zzz_RemoteConnect( coname, stack, flag,
serverlist[i], 4001))!=0)
184     {
            break;
186     }
    }
188 /* if none of the listed head nodes was reachable exit */
    if(msgbox==0)
190 {
      for(i=0; i<entries; i++)
192   {
        fprintf(stderr,"Error: RemoteConnect to host %s connection failed.\n",
194 serverlist[i]);
      }
196   destroylist(&serverlist);
      exit(EXIT_FAILURE);
198 }

200 /* we dont need that string anymore */
    free(coname);
202
    /* free server list */
204 destroylist(&serverlist);

206 /* change focus */
    ha_msgbox = zzz_Focus (msgbox, "HA");
208
    /* get sender name */
210 sender= HA_Get_Logical_Name(ha_msgbox);

212 E_init();

214 /* put together a message includeing all execution details */
    msg=mkdelmsg(getuid(), getgid(), input, argv, env, sender, path);
216 if(strlen(msg)>MAX_MSG_SIZE)
    {
218   fprintf(stderr, "Message to remote server too long. Try less STDIN.\n");
      exit(EXIT_FAILURE);
220 }

222 destroystring(&input);
    destroystring(&path);
224
    ret = zzz_VaSend(msgbox, SAFE, 0, strlen(msg)+1 , msg,
226 HEADNODEGROUP, NULL);

228 if(ret<strlen(msg))
    {
230   fprintf(stderr, "Message sent failed.\n");
      exit(EXIT_FAILURE);
232 }

234 destroystring(&msg);

236
    /* add event base for incoming messages */
238 zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
    E_sched(tout, timeout, (void *) 1);
240
```

```
  /* start event handler */
242  E_main_loop();

244  return(EXIT_SUCCESS);

246 }

248
/*****************************************************************************
250 *                                                                           *
* handle_events                                                              *
252 *                                                                           *
*****************************************************************************/
254 void handle_events(int dummy1, void *param)
{
256 /*
* local data
258 */
  /* message buffer for TRANSIS msgs */
260  char recv_buf[MAX_MSG_SIZE];
  int recv_type, amount;
262  char *out=NULL, *err=NULL;
  view *gview;
264

266  amount=zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
  /* distinguish between group chang msg and data msg */
268  /* data msg, dont care for group msgs */
  if( recv_type != VIEW_CHANGE)
270  {
    /* first recover the msgid */
272    switch(recov_id(recv_buf))
    {
274      /* message received was an response msg */
      case(RSPMSGID):
276        err=recov_stderr(recv_buf);
    out=recov_stdout(recv_buf);
278        if(out!=NULL)
        {
280          fprintf(stdout, "%s", out);
        }
282        else
    {
284      if(err!=NULL)
          {
286          fprintf(stderr, "%s", err);
        }
288    }
            destroystring(&out);
290         destroystring(&err);
        exit(EXIT_SUCCESS);
292     break;
      /* message unknown and is being ignored */
294     case(UNKNOWNMSG):
        fprintf(stderr, "Unknown message: %s\n", recv_buf);
296     exit(EXIT_FAILURE);
    }
298 }

300 }

302 void timeout(void *param)
{
```

```
304     fprintf(stderr, "Operation timed out...\n");
        exit(EXIT_FAILURE);
306
}
```

jstat.c

```
/*****************************************************************************
 * Project: JOSHUA                                                           *
 * Description: Command line tool to get queue status                        *
 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
 *                                                                           *
 *           888888                    888                                   *
 *             "88b                    888                                   *
 *              888                    888                                   *
 *              888  .d88b.  .d8888b   88888b.  888  888  8888b.              *
 *              888 d88""88b 88K       888 "88b 888  888     "88b             *
 *              888 888  888 "Y8888b.  888  888 888  888 .d888888             *
 *              88P Y88..88P      X88  888  888 Y88b 888 888  888             *
 *              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888             *
 *            .d88P                                                          *
 *          .d88P"                              2006 Kai Uhlemann            *
 *         888P"                                                             *
 *                                                                           *
 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
 * System: Linux 2.6.8-2-686-smp on i686                                     *
 *                                                                           *
 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
 *                                                                           *
*****************************************************************************/
/*****************************************************************************
 *                                                                           *
 * Headers                                                                   *
 *                                                                           *
*****************************************************************************/
#include "utils.h"
/*****************************************************************************
 *                                                                           *
 * Global data                                                               *
 *                                                                           *
*****************************************************************************/
static zzz_mbox_cap     msgbox;
/*****************************************************************************
 *                                                                           *
 * Prototypes                                                                *
 *                                                                           *
*****************************************************************************/
void handle_events(int, void *);
void timeout(void*);
/*****************************************************************************
 *                                                                           *
 * Main                                                                      *
 *                                                                           *
*****************************************************************************/
int main(int argc, char **argv, char **env)
{
/*
 * Local data
*/
  /* Transis data */
  int flag = 1;
  char *stack = NULL; /* default layer stack will be used */
```

```
56   static mbox_cap      ha_msgbox;

58   char *sender=NULL;
     char *msg=NULL;
60   char *input=NULL;
     char *path=NULL;
62   char **serverlist;
     int ret=0;
64   struct timeval tout, t_current;
     int entries=0, i=0;
66
     /* used to create username */
68   double current;
     char* coname=NULL;
70

72 #ifdef SYSCONFDIR
     char *conf = EXPAND(SYSCONFDIR);
74 #else
     char *conf = NULL;
76   fprintf(stderr, "Error: No default sysconfdir defined\n");
     exit(EXIT_FAILURE);
78 #endif

80   /* check for sane input, accept no options so far */
     if(argc>2)
82   {
       fprintf(stderr,"Error: Option not supported.\n");
84     exit(EXIT_FAILURE);
     }
86
     /* options for the config file */
88       cfg_opt_t opts[] =
     {
90
       CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
92     CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
       CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
94     CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
       CFG_STR("scheduler", "maui", CFGF_NONE),
96     CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
       CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
98     CFG_STR("job_server", "pbs_server", CFGF_NONE),
       CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
100    CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
       CFG_STR("group_com", "transis", CFGF_NONE),
102    CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
       CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
104    CFG_STR("joshua", "joshua", CFGF_NONE),
       CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
106    CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
       CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
108    /* a memory leak in libconfuse forces to leave the default to NULL */
       CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
110    CFG_END()
     };
112  cfg_t *cfg;

114  cfg = cfg_init(opts, CFGF_NONE);

116  if(cfg_parse(cfg, conf) == CFG_PARSE_ERROR)
     {
118    /* no output will be seen when uninitialized */
```

```
    fprintf(stderr, "Error: parsing configfile\n");
120   exit(EXIT_FAILURE);
  }
122
  /* alloc server list */
124 entries=cfg_size(cfg, "headnodes");
  if(entries<1)
126 {
    fprintf(stderr, "No headnode entries in config file %s\n", conf);
128   cfg_free(cfg);
    exit(EXIT_FAILURE);
130 }

132 serverlist=(char **)malloc(sizeof(char*)*(entries+1));
  if(serverlist==NULL)
134 {
    fprintf(stderr, "malloc %s\n",strerror(errno));
136   exit(EXIT_FAILURE);
  }
138
  /* servers */
140 for(i=0; i<entries; i++)
  {
142   serverlist[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
  }
144 serverlist[i]=NULL;

146
  /* free config file structure */
148 cfg_free(cfg);

150
  /* set the timeout values */
152 tout.tv_sec=10;
  tout.tv_usec=0;
154
  /* create connection name from clockticks since 1970 */
156 gettimeofday(&t_current, NULL);
  current=t_current.tv_sec*1000000.0+t_current.tv_usec;
158
  /* get the string size for the clock ticks and allocate space */
160 if((coname=(char*)malloc((snprintf(NULL, 0, "%.0lf",
  current)+1)*sizeof(char)))==NULL)
162 {
    log_err("malloc %s\n", strerror(errno));
164 }

166 /* create the connection name */
  snprintf(coname, snprintf(NULL, 0, "%.0lf", current)+1, "%.0lf",
168 current);

170 /* get the cwd */
  path=gcwd();
172
  /* get the stdin */
174 if(chkstdin()==0)
  {
176   input=readfd(STDIN_FILENO);
  }
178
  /* connect remotely to transis */
180 for(i=0; i<entries; i++)
  {
```

```
182     if((msgbox = zzz_RemoteConnect( coname , stack, flag, serverlist[i],
    4001))!=0)
184     {
          break;
186     }
    }
188 /* if none of the listed head nodes was reachable exit */
    if(msgbox==0)
190 {
      for(i=0; i<entries; i++)
192   {
        fprintf(stderr,"Error: RemoteConnect to host %s connection failed.\n",
194 serverlist[i]);
      }
196   destroylist(&serverlist);
      exit(EXIT_FAILURE);
198 }

200 /* we dont need that string anymore */
    free(coname);
202
    /* free server list */
204 destroylist(&serverlist);

206 /* change focus */
    ha_msgbox = zzz_Focus (msgbox, "HA");
208
    /* get sender name */
210 sender= HA_Get_Logical_Name(ha_msgbox);

212 E_init();

214 /* put together a message includeing all execution details */
    msg=mkstamsg(getuid(), getgid(), input, argv, env, sender, path);
216 if(strlen(msg)>MAX_MSG_SIZE)
    {
218   fprintf(stderr, "Message to remote server too long. Try less STDIN.\n");
      exit(EXIT_FAILURE);
220 }

222 destroystring(&input);
    destroystring(&path);
224
    ret = zzz_VaSend(msgbox, SAFE, 0, strlen(msg)+1 , msg,
226 HEADNODEGROUP, NULL);

228 if(ret<strlen(msg))
    {
230   fprintf(stderr, "Message sent failed.\n");
      exit(EXIT_FAILURE);
232 }

234 destroystring(&msg);


236
    /* add event base for incoming messages */
238 zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
    E_sched(tout, timeout, (void *) 1);
240
    /* start event handler */
242 E_main_loop();

244 return(EXIT_SUCCESS);
```

```
246 }

248
/*****************************************************************************
250 *                                                                           *
* handle_events                                                             *
252 *                                                                           *
*****************************************************************************/
254 void handle_events(int dummy1, void *param)
{
256 /*
* local data
258 */
  /* message buffer for TRANSIS msgs */
260   char recv_buf[MAX_MSG_SIZE];
  int recv_type, amount;
262   char *out=NULL, *err=NULL;
  view *gview;
264

266   amount=zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
  /* distinguish between group chang msg and data msg */
268   /* data msg, dont care for group msgs */
  if( recv_type != VIEW_CHANGE)
270   {
    /* first recover the msgid */
272     switch(recov_id(recv_buf))
    {
274       /* message received was an response msg */
      case(RSPMSGID):
276         err=recov_stderr(recv_buf);
  out=recov_stdout(recv_buf);
278         if(out!=NULL)
        {
280           fprintf(stdout, "%s", out);
        }
282         else
  {
284     if(err!=NULL)
          {
286           fprintf(stderr, "%s", err);
          }
288   }
          destroystring(&out);
290         destroystring(&err);
          exit(EXIT_SUCCESS);
292         break;
      /* message unknown and is being ignored */
294       case(UNKNOWNMSG):
          fprintf(stderr, "Unknown message: %s\n", recv_buf);
296         exit(EXIT_FAILURE);
    }
298   }

300 }

302 void timeout(void *param)
{
304     fprintf(stderr, "Operation timed out...\n");
    exit(EXIT_FAILURE);
306

}
```

A.3.2 jmutex

jmutex.c

```
/*****************************************************************************
 * Project: JOSHUA                                                           *
 * Description: JOSHUA mutex to request job execution on cluster node        *
 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
 *                                                                           *
 *           888888                    888                                   *
 *             "88b                    888                                   *
 *              888                    888                                   *
 *              888  .d88b.  .d8888b   88888b.  888  888  8888b.              *
 *              888 d88""88b 88K       888 "88b 888  888     "88b            *
 *              888 888  888 "Y8888b.  888  888 888  888 .d888888            *
 *              88P Y88..88P      X88  888  888 Y88b 888 888  888            *
 *              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888            *
 *            .d88P                                                          *
 *          .d88P"                              2006 Kai Uhlemann            *
 *         888P"                                                             *
 *                                                                           *
 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
 * System: Linux 2.6.8-2-686-smp on i686                                     *
 *                                                                           *
 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
 *                                                                           *
 *****************************************************************************/
/*****************************************************************************
 *                                                                           *
 * Headers                                                                   *
 *                                                                           *
 *****************************************************************************/
#include "utils.h"
/*****************************************************************************
 *                                                                           *
 * Global data                                                               *
 *                                                                           *
 *****************************************************************************/
static zzz_mbox_cap      msgbox;
/*****************************************************************************
 *                                                                           *
 * Prototypes                                                                *
 *                                                                           *
 *****************************************************************************/
void handle_events(int, void *);
void timeout(void*);
/*****************************************************************************
 *                                                                           *
 * Main                                                                      *
 *                                                                           *
 *****************************************************************************/
int main(int argc, char **argv, char **env)
{
/*
 * Local data
 */
  /* Transis data */
  int flag = 1;
  char *stack = NULL; /* default layer stack will be used */
  static mbox_cap      ha_msgbox;
```

```
58   char *sender=NULL;
     char *msg=NULL;
60   char **serverlist;
     int ret=0;
62   struct timeval tout, t_current;
     int entries=0, i=0;
64
     /* used to create username */
66   double current;
     char* coname=NULL;
68


70
#ifdef SYSCONFDIR
72   char *conf = EXPAND(SYSCONFDIR);
#else
74   char *conf = NULL;
     fprintf(stderr, "No default sysconfdir defined\n");
76   exit(EXIT_FAILURE);
#endif
78   /* options for the config file */
         cfg_opt_t opts[] =
80   {

82     CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
       CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
84     CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
       CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
86     CFG_STR("scheduler", "maui", CFGF_NONE),
       CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
88     CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
       CFG_STR("job_server", "pbs_server", CFGF_NONE),
90     CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
       CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
92     CFG_STR("group_com", "transis", CFGF_NONE),
       CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
94     CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
       CFG_STR("joshua", "joshua", CFGF_NONE),
96     CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
       CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
98     CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
       /* a memory leak in libconfuse forces to leave the default to NULL */
100    CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
       CFG_END()
102  };
     cfg_t *cfg;
104
     cfg = cfg_init(opts, CFGF_NONE);
106
     if(cfg_parse(cfg, conf) == CFG_PARSE_ERROR)
108  {
       /* no output will be seen when uninitialized */
110    fprintf(stderr, "Error: parsing configfile\n");
       exit(EXIT_FAILURE);
112  }

114  /* alloc server list */
     entries=cfg_size(cfg, "headnodes");
116  if(entries<1)
     {
118    fprintf(stderr, "No headnode entries in config file %s\n", conf);
       cfg_free(cfg);
120    exit(EXIT_FAILURE);
```

```
  }

  serverlist=(char **)malloc(sizeof(char*)*(entries+1));
  if(serverlist==NULL)
  {
    fprintf(stderr, "malloc %s\n",strerror(errno));
    exit(EXIT_FAILURE);
  }

  /* servers */
  for(i=0; i<entries; i++)
  {
    serverlist[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
  }
  serverlist[i]=NULL;


  /* free config file structure */
  cfg_free(cfg);


  /* set the timeout values */
  tout.tv_sec=5;
  tout.tv_usec=5;


  /* create connection name from clockticks since 1970 */
  gettimeofday(&t_current, NULL);
  current=t_current.tv_sec*1000000.0+t_current.tv_usec;

  /* get the string size for the clock ticks and allocate space */
  if((coname=(char*)malloc((snprintf(NULL, 0, "%.0lf",
current)+1)*sizeof(char)))==NULL)
  {
    log_err("malloc %s\n", strerror(errno));
  }

  /* create the connection name */
  snprintf(coname, snprintf(NULL, 0, "%.0lf", current)+1, "%.0lf",
current);

  /* connect remotely to transis */
  for(i=0; i<entries; i++)
  {
    if((msgbox = zzz_RemoteConnect( coname , stack , flag ,
serverlist[i], 4001))!=0)
    {
      break;
    }
  }
  /* if none of the listed head nodes was reachable exit */
  if(msgbox==0)
  {
    for(i=0; i<entries; i++)
    {
      fprintf(stderr,"Error: RemoteConnect to host %s connection\
failed.\n", serverlist[i]);
    }
    destroylist(&serverlist);
    exit(EXIT_FAILURE);
  }

  /* we dont need that string anymore */
```

```
184   free(coname);

186   /* free server list */
      destroylist(&serverlist);
188
      /* change focus */
190   ha_msgbox = zzz_Focus (msgbox, "HA");

192   /* get sender name */
      sender= HA_Get_Logical_Name(ha_msgbox);
194
      E_init();
196
      /* put together a message includeing all execution details */
198   msg=mkstrmsg(atoi(argv[1]), sender);
      if(strlen(msg)>MAX_MSG_SIZE)
200   {
        fprintf(stderr, "Message to remote server too long.\n");
202     exit(EXIT_FAILURE);
      }
204

206   ret = zzz_VaSend(msgbox, SAFE, 0, strlen(msg)+1 , msg,
HEADNODEGROUP, NULL);
208
      if(ret<strlen(msg))
210   {
        fprintf(stderr, "Message sent failed.\n");
212     exit(EXIT_FAILURE);
      }
214
      destroystring(&msg);
216

218   /* add event base for incoming messages */
      zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
220 //  E_sched(tout, timeout, (void *) 1);

222   /* start event handler */
      E_main_loop();
224
      return(EXIT_SUCCESS);
226
}
228

230 /*****************************************************************************
*                                                                            *
232 * handle_events                                                              *
*                                                                            *
234 *****************************************************************************/
void handle_events(int dummy1, void *param)
236 {
/*
238 * local data
*/
240   /* message buffer for TRANSIS msgs */
      char recv_buf[MAX_MSG_SIZE];
242   int recv_type, amount;
      view *gview;
244

246   amount=zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
```

```
  /* distinguish between group chang msg and data msg */
248 /* data msg, dont care for group msgs */
  if( recv_type != VIEW_CHANGE)
250 {
    /* first recover the msgid */
252 switch(recov_id(recv_buf))
    {
254   /* message received was an response msg */
      case(STRMSGID):
256     fprintf(stderr, "Allowed\n");
        exit(EXIT_PBS_SUCCESS);
258     break;
      case(FNSMSGID):
260     fprintf(stderr, "Not Allowed\n");
        exit(EXIT_PBS_ABORT);
262     break;
      /* message unknown and is being ignored */
264   case(UNKNOWNMSG):
        fprintf(stderr, "Unknown message: %s\n", recv_buf);
266     exit(EXIT_PBS_ABORT);
    }
268 }

270 }

272 void timeout(void *param)
  {
274   fprintf(stderr, "Operation timed out...\n");
    exit(EXIT_FAILURE);
276
  }
```

jjdone.c

```
  /*****************************************************************************
2 * Project: JOSHUA                                                           *
  * Description: JOSHUA mutex to signal job finished on cluster node           *
4 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
  *                                                                           *
6 *           888888                    888                                   *
  *             "88b                    888                                   *
8 *              888                    888                                   *
  *              888  .d88b.  .d8888b   88888b.  888  888  8888b.              *
10 *             888 d88""88b 88K       888 "88b 888  888     "88b            *
  *              888 888  888 "Y8888b.  888  888 888  888 .d888888            *
12 *             88P Y88..88P      X88  888  888 Y88b 888 888  888            *
  *              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888            *
14 *            .d88P                                                         *
  *          .d88P"                             2006 Kai Uhlemann            *
16 *         888P"                                                           *
  *                                                                           *
18 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
  * System: Linux 2.6.8-2-686-smp on i686                                     *
20 *                                                                           *
  * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
22 *                                                                           *
  *****************************************************************************/
24 /*****************************************************************************
  *                                                                           *
26 * Headers                                                                   *
  *                                                                           *
28 *****************************************************************************/
```

```
#include "utils.h"
/*****************************************************************************
*                                                                            *
* Global data                                                                *
*                                                                            *
*****************************************************************************/
static zzz_mbox_cap      msgbox;
/*****************************************************************************
*                                                                            *
* Main                                                                       *
*                                                                            *
*****************************************************************************/
int main(int argc, char **argv, char **env)
{
/*
* Local data
*/
  /* Transis data */
  int flag = 1;
  char *stack = NULL; /* default layer stack will be used */
  static mbox_cap      ha_msgbox;

  char *sender=NULL;
  char *msg=NULL;
  char **serverlist;
  int ret=0;
  struct timeval tout;
  int entries=0, i=0;

#ifdef SYSCONFDIR
  char *conf = EXPAND(SYSCONFDIR);
#else
  char *conf = NULL;
  fprintf(stderr, "No default sysconfdir defined\n");
  exit(EXIT_FAILURE);
#endif
  /* options for the config file */
      cfg_opt_t opts[] =
  {

    CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
    CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
    CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
    CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
    CFG_STR("scheduler", "maui", CFGF_NONE),
    CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
    CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
    CFG_STR("job_server", "pbs_server", CFGF_NONE),
    CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
    CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
    CFG_STR("group_com", "transis", CFGF_NONE),
    CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
    CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
    CFG_STR("joshua", "joshua", CFGF_NONE),
    CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
    CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
    CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
    /* a memory leak in libconfuse forces to leave the default to NULL */
    CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
    CFG_END()
  };
  cfg_t *cfg;
```

```
92  cfg = cfg_init(opts, CFGF_NONE);

94  if(cfg_parse(cfg, conf) == CFG_PARSE_ERROR)
    {
96    /* no output will be seen when uninitialized */
      fprintf(stderr, "Error: parsing configfile\n");
98    exit(EXIT_FAILURE);
    }
100
    /* alloc server list */
102 entries=cfg_size(cfg, "headnodes");
    if(entries<1)
104 {
      fprintf(stderr, "No headnode entries in config file %s\n", conf);
106   cfg_free(cfg);
      exit(EXIT_FAILURE);
108 }

110 serverlist=(char **)malloc(sizeof(char*)*(entries+1));
    if(serverlist==NULL)
112 {
      fprintf(stderr, "malloc %s\n",strerror(errno));
114   exit(EXIT_FAILURE);
    }
116
    /* servers */
118 for(i=0; i<entries; i++)
    {
120   serverlist[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
    }
122 serverlist[i]=NULL;

124
    /* free config file structure */
126 cfg_free(cfg);

128
    /* set the timeout values */
130 tout.tv_sec=5;
    tout.tv_usec=5;
132
    /* connect remotely to transis */
134 for(i=0; i<entries; i++)
    {
136   if((msgbox = zzz_RemoteConnect("jdone", stack, flag,
serverlist[i], 4001))!=0)
138   {
        break;
140   }
    }
142 /* if none of the listed head nodes was reachable exit */
    if(msgbox==0)
144 {
      for(i=0; i<entries; i++)
146   {
        fprintf(stderr,"Error: RemoteConnect to host %s connection\
148 failed.\n", serverlist[i]);
      }
150   destroylist(&serverlist);
      exit(EXIT_FAILURE);
152 }

154 /* free server list */
```

```
    destroylist(&serverlist);
156
    /* change focus */
158 ha_msgbox = zzz_Focus (msgbox, "HA");

160 /* get sender name */
    sender= HA_Get_Logical_Name(ha_msgbox);
162
    E_init();
164
    /* put together a message includeing all execution details */
166 msg=mkfnsmsg(atoi(argv[1]), sender);
    if(strlen(msg)>MAX_MSG_SIZE)
168 {
      fprintf(stderr, "Message to remote server too long.\n");
170   exit(EXIT_FAILURE);
    }
172
    ret = zzz_VaSend(msgbox, SAFE, 0, strlen(msg)+1 , msg,
174 HEADNODEGROUP, NULL);

176 if(ret<strlen(msg))
    {
178   fprintf(stderr, "Message sent failed.\n");
      exit(EXIT_FAILURE);
180 }

182 destroystring(&msg);


184
    return(EXIT_SUCCESS);
186
}
```

Prologue script

```
1 #!/bin/sh
  ##############################################################################
3 # Project: JOSHUA                                                            #
  # Description: proloque script for cluster mutex                             #
5 # Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             #
  #                                                                            #
7 #           888888                    888                                  #
  #             "88b                    888                                  #
9 #              888                    888                                  #
  #              888  .d88b.  .d8888b  88888b.  888  888  8888b.              #
11 #             888 d88""88b 88K      888 "88b 888  888     "88b             #
  #              888 888  888 "Y8888b. 888  888 888  888 .d888888             #
13 #              88P Y88..88P      X88 888  888 Y88b 888 888  888             #
  #              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             #
15 #            .d88P                                                         #
  #          .d88P"                            2006 Kai Uhlemann             #
17 #        888P"                                                             #
  #                                                                            #
19 # Created at: Mon Nov  7 10:58:14 EST 2005                                 #
  # System: Linux 2.6.8-2-686-smp on i686                                      #
21 #                                                                           #
  # Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      #
23 #                                                                           #
  ##############################################################################
25
  echo "Prologue Args:" >/tmp/pre.txt
```

```
echo "Job ID: $1">>/tmp/pre.txt
echo "User ID: $2">>/tmp/pre.txt
echo "Group ID: $3">>/tmp/pre.txt

echo "Starting jutex">>/tmp/pre.txt
/home/kai/joshua-0.1/jmutex/jmutex $1
EXIT_CODE=$?

echo -n "Exiting with exit code $EXIT_CODE...">>/tmp/pre.txt
echo "done.">>/tmp/pre.txt
exit $EXIT_CODE

exit 0
```

Epilogue script

```
#!/bin/sh
################################################################################
# Project: JOSHUA                                                              #
# Description: epiloque script for cluster mutex                               #
# Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                               #
#                                                                              #
#           888888                    888                                      #
#             "88b                    888                                      #
#              888                    888                                      #
#              888  .d88b.  .d8888b   88888b.  888  888  8888b.                 #
#              888 d88""88b 88K       888 "88b 888  888     "88b              #
#              888 888  888 "Y8888b.  888  888 888  888 .d888888              #
#              88P Y88..88P      X88  888  888 Y88b 888 888  888              #
#              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888              #
#            .d88P                                                             #
#          .d88P"                             2006 Kai Uhlemann              #
#         888P"                                                                #
#                                                                              #
# Created at: Mon Nov  7 10:58:14 EST 2005                                     #
# System: Linux 2.6.8-2-686-smp on i686                                        #
#                                                                              #
# Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.        #
#                                                                              #
################################################################################

echo "Prologue Args:" >/tmp/epi.txt
echo "Job ID: $1">>/tmp/epi.txt
echo "User ID: $2">>/tmp/epi.txt
echo "Group ID: $3">>/tmp/epi.txt
echo "JOB name : $4">>/tmp/epi.txt

echo "Releasing Job $4...">>/tmp/epi.txt
/home/kai/joshua-0.1/jmutex/jjdone $1
echo "done.">>/tmp/epi.txt

exit 0
```

A.3.3 joshua

jinit.c

```
/*****************************************************************************
 * Project: JOSHUA                                                           *
 * Description: JOSHUA SERVER daemon startup init                            *
 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
 *                                                                           *
 *           888888                    888                                   *
 *             "88b                    888                                   *
 *              888                    888                                   *
 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.               *
 *              888 d88""88b 88K      888 "88b 888  888     "88b              *
 *              888 888  888 "Y8888b. 888  888 888  888 .d888888              *
 *              88P Y88..88P      X88 888  888 Y88b 888 888  888              *
 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888              *
 *            .d88P                                                          *
 *          .d88P"                             2006 Kai Uhlemann             *
 *         888P"                                                             *
 *                                                                           *
 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
 * System: Linux 2.6.8-2-686-smp on i686                                     *
 *                                                                           *
 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
 *                                                                           *
*****************************************************************************/
/*****************************************************************************
 *                                                                           *
 * Headers                                                                   *
 *                                                                           *
*****************************************************************************/
#include "utils.h"
#include "startup.h"
#include "signals.h"
#define JOBS "job scheduler............."
#define RESM "resource manager........."
#define JOSH "joshua..................."
#define OBSE "jobserver................"
#define DONE " done\n"
#define ILEN 128
/*****************************************************************************
 *                                                                           *
 * Global data                                                               *
 *                                                                           *
*****************************************************************************/
extern char *program_name;
extern char *configfile;
extern srvdata svdat;
extern FILE *log_target_err;
extern FILE *log_target_out;
void jobserver(int pipefd);
/*****************************************************************************
 *                                                                           *
 * Main                                                                      *
 * argc - argument counter                                                   *
 * argv - argument vector                                                    *
 * env  - environment vector                                                 *
 *                                                                           *
*****************************************************************************/
int main(int argc, char **argv)
{
/*
 * local data
 */
  int i=0;
  int p0[2];
```

```
64  int fd[2];
    pid_t pid;
66  int initpipe[2];
    char *initout=NULL;
68   /* data for select */
    fd_set fdset;
70  struct timeval tv;
    int ret=0;
72  int initcheck=0;

74  program_name = argv[0];
    decode_switches (argc, argv);
76
    /* init init pipe */
78  if(pipe(initpipe)!=0)
    {
80    fprintf(stderr,"pipe %s\n",strerror(errno));
      log_err("pipe %s\n",strerror(errno));
82  }

84  /* daemonize me */
    i=fork();
86  if (i<0) {exit(EXIT_FAILURE);} /* fork error */
    if (i>0) {
88          /* close write for init pipe */
            close(initpipe[1]);
90          fprintf(stdout, "jinit started...\nAttemting to start JOSHUA components...\n");
            /* decide which output to read */
92          /* empty fdset */
            FD_ZERO(&fdset);
94          FD_SET(initpipe[0], &fdset);
            /* wait 5s for input */
96          tv.tv_sec = 12;
            tv.tv_usec = 0;
98          ret=1;

100         /* check max file escriptor */
            ret=select(initpipe[0]+1,&fdset,NULL,NULL,&tv);
102         if(ret==-1)
            {
104           fprintf(stderr,"select %s\n", strerror(errno));
            }
106         else
            {
108           if(ret>0)
              {
110             initout=readfd(initpipe[0]);
              }
112           else
              {
114             log_warn("select timeout\n");
                fprintf(stderr,"select timeout\n");
116           }
            }
118         fprintf(stdout, "%s", initout);
            fflush(NULL);
120         deletedata(&svdat);
            if(initout!=NULL)
122         {
              initcheck=strlen(initout);
124         }
            if(initcheck==ILEN)
126         {
```

```
            destroystring(&initout);
128         exit(EXIT_SUCCESS);
          }
130       else
          {
132         fprintf(stderr," failed.\nCheck logs for further information.\n");
            destroystring(&initout);
134         exit(EXIT_FAILURE);
          }
136
          destroystring(&initout);
138       exit(EXIT_SUCCESS);
    } /* parent exits */
140
  /* child (daemon) continues */
142 /* its always safe to do that */
  initdata(&svdat, configfile);
144 log_target_out=fopen(svdat.logfile, "a");
  log_target_err=fopen(svdat.errlog,"a");
146
  bootinit(log_target_out, log_target_err);
148
  if(log_target_out==NULL)
150 {
    log_err("logfile %s: %s\n",svdat.logfile ,strerror(errno));
152 }
  if(log_target_err==NULL)
154 {
    log_err("errorlog %s: %s\n", svdat.errlog,strerror(errno));
156 }

158 /* make me a daemon */
  fd[0]=open(svdat.logfile, O_WRONLY|O_CREAT|O_APPEND,S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH
      );
160 fd[1]=open(svdat.errlog,  O_WRONLY|O_CREAT|O_APPEND,S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH
      );

162 dup2(fd[0], STDOUT_FILENO);
  dup2(fd[1], STDERR_FILENO);
164
  close(STDIN_FILENO);
166 /*close read for init pipe */
  close(initpipe[0]);
168
  log_info("jbootup started...\n");
170 writefd(initpipe[1], JOBS, strlen(JOBS));
  /* create maui process */
172 pid=fork();
  switch(pid)
174 {
    /* error */
176   case -1:  log_err("fork %s\n",strerror(errno));
              break;
178   /* child */
      case 0:   log_warn("maui started with %s %s %s\n", svdat.watch[0].prgexec, svdat.watch[0]
          .name, "-d");
180           /* close the init pipe */
              close(initpipe[1]);
182           execl(svdat.watch[0].prgexec, svdat.watch[0].name, "-d","0",  NULL);
              exit(-1);
184           break;
    /* parent */
186   default:  /* reset observer pid */
```

```
                svdat.watch[0].pid=pid;
188             break;

190     }
        log_warn("maui started\n");
192     writefd(initpipe[1], DONE, strlen(DONE));

194
        sigsyncinit();
196     /* prepare bidirectional pipe for data interchange with observer */
        if(pipe(p0)!=0)
198     {
          log_err("pipe %s\n",strerror(errno));
200     }

202     writefd(initpipe[1], JOSH, strlen(JOSH));

204     /* create joshua process */
        pid=fork();
206     switch(pid)
        {
208       /* error */
          case -1:  log_err("fork %s\n",strerror(errno));
210               break;
          /* child */
212       case 0:   log_info("joshua started\n");
                  /* close the init pipe */
214               close(initpipe[1]);
                  execl(svdat.watch[3].prgexec, svdat.watch[3].name, "-c", svdat.watch[3].conf,
                      NULL);
216               exit(-1);
                  break;
218       /* parent */
          default:  /* reset observer pid */
220               svdat.watch[3].pid=pid;
                  break;
222     }

224     log_info("Waiting till joshua is ready..\n");
        waitforsig();
226     writefd(initpipe[1], DONE, strlen(DONE));

228     writefd(initpipe[1], RESM, strlen(RESM));
        /* create pbs_server process */
230     pid=fork();
        switch(pid)
232     {
          /* error */
234       case -1:  log_err("fork %s\n",strerror(errno));
                  break;
236       /* child */
          case 0:   log_info("pbs_server started\n");
238               /* close the init pipe */
                  close(initpipe[1]);
240               char *envs[]={"PBSDEBUG=1", NULL};
                  execle(svdat.watch[1].prgexec, svdat.watch[1].name, NULL, envs);
242               exit(-1);
                  break;
244       /* parent */
          default:  /* reset observer pid */
246               svdat.watch[1].pid=pid;
                  break;
248     }
```

```
250   Sleep(1,0);
      log_info("Sending signal that pbs is running\n");
252   notify_joshua(svdat.watch[3].pid);
      log_info("Signal send\n");
254
      sigsyncunset();
256
      writefd(initpipe[1], DONE, strlen(DONE));
258   writefd(initpipe[1], OBSE, strlen(OBSE));

260   /* create observer process */
      pid=fork();
262   switch(pid)
      {
264     /* error */
        case -1:  log_err("fork %s\n",strerror(errno));
266               break;
        /* child */
268     case 0:   log_info("jobserver started\n");
                  /* close the init pipe */
270               close(initpipe[1]);
                  /* close read for first pipe */
272               close(p0[0]);
                  jobserver(p0[1]);
274               break;
        /* parent */
276     default:  /* close write for first pipe */
                  close(p0[1]);
278               /* reset observer pid */
                  svdat.observer=pid;
280               break;
      }
282
      writefd(initpipe[1], DONE, strlen(DONE));
284   /* close the init pipe */
      close(initpipe[1]);
286
      log_warn("Set to pause()\n");
288   pause();
      log_warn("Over pause()\n");
290
      shutd("jbootup ended");
292   return EXIT_SUCCESS;
}
294 /*****************************************************************************
*                                                                            *
296 * jobserver                                                                  *
*                                                                            *
298 *****************************************************************************/
void jobserver(int pipefd)
300 {
      char buf[10*4096]={0};
302   int re=0;
      long max=0;
304   max=fpathconf(pipefd, _PC_PIPE_BUF);
      if(max==-1)
306   {
        max=10*4096;
308   }
      /* reset observer pid */
310   svdat.observer=getpid();
```

```
312   log_warn("Write to pipe %d bytes maximal possible\n", fpathconf(pipefd,
      _PC_PIPE_BUF));
314   /* block with pipe  */
      re=write(pipefd, &buf, max+1);
316   log_warn("done with pipe %d bytes written\n", re);
      shutd("jobserver ended...");
318   exit(EXIT_SUCCESS);
    }
```

joshua.c

```
    /*****************************************************************************
2   * Project: JOSHUA                                                           *
    * Description: JOSHUA SERVER daemon                                         *
4   * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
    *                                                                           *
6   *           888888                   888                                  *
    *             "88b                   888                                  *
8   *              888                   888                                  *
    *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
10  *              888 d88""88b 88K      888 "88b 888  888     "88b             *
    *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
12  *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
    *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
14  *            .d88P                                                          *
    *          .d88P"                           2006 Kai Uhlemann             *
16  *         888P"                                                           *
    *                                                                           *
18  * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
    * System: Linux 2.6.8-2-686-smp on i686                                     *
20  *                                                                           *
    * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
22  *                                                                           *
    *****************************************************************************/
24  /*****************************************************************************
    *                                                                           *
26  * Headers                                                                   *
    *                                                                           *
28  *****************************************************************************/
    #include "utils.h"
30  #include "server.h"
    #include "startup.h"
32  #include "signals.h"
    /*****************************************************************************
34  *                                                                           *
    * Global data                                                               *
36  *                                                                           *
    *****************************************************************************/
38  /* Transis relevant message boxes */
    extern zzz_mbox_cap msgbox;
40  extern zzz_mbox_cap msgbox_join;
    extern char *program_name;
42  extern char *configfile;
    extern srvdata svdat;
44  extern FILE *log_target_err;
    extern FILE *log_target_out;
46  /*****************************************************************************
    *                                                                           *
48  * Main                                                                      *
    * argc - argument counter                                                   *
50  * argv - argument vector                                                    *
    * env  - environment vector                                                 *
```

```
52 *                                                                      *
   ************************************************************************/
54 int main(int argc, char **argv)
   {
56 /*
   * local data
58 */
     /* default TRANSIS layer stack will be used */
60   int flag = 1;
     char *stack = NULL;
62   mbox_cap ha_msgbox;
     mbox_cap ha_msgbox_join;
64
     /* init server */
66   program_name = argv[0];
     decode_switches (argc, argv);
68   initdata(&svdat, configfile);
     log_target_out=fopen(svdat.logfile, "a");
70   log_target_err=fopen(svdat.errlog,"a");

72
     /* its always safe to do that */
74   serverinit(log_target_out, log_target_err);

76   if(log_target_out==NULL)
     {
78     log_err("logfile %s: %s\n",svdat.logfile ,strerror(errno));
     }
80   if(log_target_err==NULL)
     {
82     log_err("errorlog %s: %s\n", svdat.errlog,strerror(errno));
     }
84   log_info("I'm there\n");

86   /* init msgbox for join operation */
     msgbox_join = zzz_Connect("join", stack, flag);
88

90   /* set focus */
     ha_msgbox_join = zzz_Focus (msgbox_join, "HA");
92   zzz_Join(msgbox_join, "join");

94   /* open transis conection */
     msgbox = zzz_Connect("headmaster", stack, flag);
96
     /* we dont need that string anymore */
98   /*free(coname);*/

100  if (msgbox == NULL)
     {
102    log_err("Connection to TRANSIS failed\n");
     }
104  /* set focus */
     ha_msgbox = zzz_Focus (msgbox, "HA");
106
     E_init();
108
     /* join the headnode group */
110  zzz_Join(msgbox, HEADNODEGROUP);

112  /* add event base for incoming messages */
     zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
114
```

```
    log_info("JOSHUA daemon started...\n");
116
    /* start event handler */
118 E_main_loop();

120
    return EXIT_SUCCESS;
122 }
```

server.h

```
    /*************************************************************************
2   * Project: JOSHUA                                                       *
    * Description: functions for the JOSHUA server                          *
4   * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                        *
    *                                                                       *
6   *           888888                   888                              *
    *             "88b                   888                              *
8   *              888                   888                              *
    *              888  .d88b.  .d8888b  88888b.  888  888  8888b.          *
10  *              888 d88""88b 88K      888 "88b 888  888     "88b         *
    *              888 888  888 "Y8888b. 888  888 888  888 .d888888         *
12  *              88P Y88..88P      X88 888  888 Y88b 888 888  888         *
    *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888         *
14  *            .d88P                                                      *
    *          .d88P"                           2006 Kai Uhlemann          *
16  *         888P"                                                         *
    *                                                                       *
18  * Created at: Mon Nov  7 10:58:14 EST 2005                              *
    * System: Linux 2.6.8-2-686-smp on i686                                 *
20  *                                                                       *
    * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved. *
22  *                                                                       *
    *************************************************************************/
24  /*************************************************************************
    *                                                                       *
26  * handle_events                                                         *
    *                                                                       *
28  *************************************************************************/
    /*!
30   \fn void handle_events(int dummy1, void *param);
     \brief function handles incomming events for TRANSIS messages
32   handle_events returns nothing
     \param dummy unused
34   \param *param unused
    */
36  void        handle_events(int, void *);
    /*************************************************************************
38  *                                                                       *
    * handle_exec                                                           *
40  *                                                                       *
    *************************************************************************/
42  /*!
     \fn void handle_join(int dummy1, void *param);
44   \brief function handles incomming events for TRANSIS messages during join event
     handle_join returns nothing
46   \param dummy unused
     \param *param unused
48  */
    void handle_join(int dummy1, void *param);
50  /*************************************************************************
    *                                                                       *
```

```
52 * handle_exec                                                              *
   *                                                                          *
54 ****************************************************************************/
   /*!
56  \fn void handle_exec(char *msg, char *cmd, int identifier);
    \brief function handles the add/summit/del/stat message events
58  function e.g. performs the actual submission of jobs using the stored stdin,
    arg and environment from a message
60  handle_exec returns nothing
    \param *msg message string transmitted via TRANIS
62  \param *cmd command to execute
    \param identifier message identifier
64 */
   void handle_exec(char *msg, char *cmd, int identifier);
66 /****************************************************************************
   *                                                                          *
68 * handle_jutex                                                             *
   *                                                                          *
70 ****************************************************************************/
   /*!
72  \fn void handle_jutex(char *msg, char *cmd, int identifier);
    \brief function handles the start and finish message events
74   handle_jutex returns nothing
    \param *msg message string transmitted via TRANIS
76  \param identifier message identifier
   */
78 void handle_jutex(char *msg, int identifier);
   /****************************************************************************
80 *                                                                          *
   * checkexec                                                                *
82 *                                                                          *
   ****************************************************************************/
84 /*!
    \fn int checkexec(int ldone, int lsub, int gendone, int gensub, int jid);
86  \brief function to check start message for sanity
    checkexec returns 0 on success or -1 in case of failure
88  \param ldone last job done
    \param lsub last job submitted
90  \param gendone generation counter for jobs done
    \param gensub generation counter for jobs submitted
92  \param jid currend job identifier to check
   */
94 int checkexec(int ldone, int lsub, int gendone, int gensub, int jid);
   /****************************************************************************
96 *                                                                          *
   * do_join                                                                  *
98 *                                                                          *
   ****************************************************************************/
100 /*!
    \fn void do_join(void);
102 \brief function handles join event for new member
    do_join returns nothing
104 */
   void do_join(void);
106 /****************************************************************************
   *                                                                          *
108 * assist_join                                                              *
   *                                                                          *
110 ****************************************************************************/
   /*!
112 \fn void assist_join(void);
    \brief function handles join event for existing member
114 assist_join returns nothing
```

```
*/
void assist_join(void);
/*****************************************************************************
*                                                                            *
* handle_join                                                                *
*                                                                            *
*****************************************************************************/
/*!
 \fn void handle_join(char *msg, char *cmd, int identifier);
 \brief function handles the add/summit/del/stat/start and finish message events
 function e.g. performs the actual submission of jobs using the stored stdin,
 arg and environment from a message
 handle_join returns nothing
 \param *msg message string transmitted via TRANIS
 \param *cmd command to execute
 \param identifier message identifier
*/
void handle_join(int dummy1, void *param);
```

server.c

```
/*****************************************************************************
* Project: JOSHUA                                                            *
* Description: functions for the JOSHUA server                               *
* Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
*                                                                            *
*           888888                    888                                  *
*             "88b                    888                                  *
*              888                    888                                  *
*              888  .d88b.  .d8888b   88888b.  888  888  8888b.             *
*              888 d88""88b 88K       888 "88b 888  888     "88b            *
*              888 888  888 "Y8888b.  888  888 888  888 .d888888            *
*              88P Y88..88P      X88  888  888 Y88b 888 888  888            *
*              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888            *
*            .d88P                                                           *
*          .d88P"                              2006 Kai Uhlemann             *
*         888P"                                                              *
*                                                                            *
* Created at: Mon Nov  7 10:58:14 EST 2005                                   *
* System: Linux 2.6.8-2-686-smp on i686                                      *
*                                                                            *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
*                                                                            *
*****************************************************************************/
/*****************************************************************************
*                                                                            *
* Headers                                                                    *
*                                                                            *
*****************************************************************************/
#include "utils.h"
#include "startup.h"
#include "server.h"
#include "signals.h"
/*****************************************************************************
*                                                                            *
* Global data                                                                *
*                                                                            *
*****************************************************************************/
zzz_mbox_cap msgbox;
zzz_mbox_cap msgbox_join;
extern srvdata svdat;
char *firstone=NULL;
```

```
long members=1;
int master=0;
int joined=0;
/* last submitted job id */
static int lsub=-1;
/* last job done id */
static int ldone=-1;
/* submission generation counter for turnover */
static int gensub=0;
/* finished job generation counter for turnover */
static int gendone=0;
/* -1 means no job runnung or submitted, yet */
/*****************************************************************************
*                                                                            *
* handle_events                                                              *
*                                                                            *
*****************************************************************************/
void handle_events(int dummy1, void *param)
{
/*
* local data
*/
  /* message buffer for TRANSIS msgs */
  char recv_buf[MAX_MSG_SIZE];
  char *myname;
  int i, recv_type, amount;
  view *gview;
  static mbox_cap      ha_msgbox;


  amount=zzz_Receive(msgbox, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
  log_info("++++ Event ++++ Message received ++++ size = %d\n", amount);
  /* distinguish between gtoup chang msg and data msg */
  /* data msg */
  if( recv_type != VIEW_CHANGE)
  {
    /* first recover the msgid */
    switch(recov_id(recv_buf))
    {
      /* message received was an submit msg*/
      case(ADDMSGID): handle_exec(recv_buf, svdat.submit, ADDMSGID);
                      break;
      case(STAMSGID): handle_exec(recv_buf, svdat.stat, STAMSGID);
                      break;
      case(DELMSGID): handle_exec(recv_buf, svdat.del, DELMSGID);
                      break;
      case(STRMSGID): handle_jutex(recv_buf, STRMSGID);
                      break;
      case(FNSMSGID): handle_jutex(recv_buf, FNSMSGID);
                      break;
     /* message unknown and is being ignored */
      case(UNKNOWNMSG): log_warn("Unknown message: %s\n", recv_buf);

    }
  }
  /* group change msg */
  else
  { /* only headnode group membership changes are interesting */
    if(strcmp(HEADNODEGROUP, gview->members[0])!=0)
    {
      return;
    }
    log_info("++ Group change in group %s from %lu to %lu member(s)\n",
```

```
105         gview->members[0], members, gview->nmembers);

107
    /* change focus */
109 ha_msgbox = zzz_Focus (msgbox, "HA");

111 /* get own name */
    myname= HA_Get_Logical_Name(ha_msgbox);
113
    /* set master to the first one if none set, yet */
115 if(firstone==NULL)
    {
117   /* start allocating */
      firstone = (char *)malloc(sizeof(char)*(MBOX_NAME_LEN+1));
119   if(firstone==NULL)
      {
121     log_err("malloc %s\n",strerror(errno));
      }
123
      /* intit used char fields */
125   memset(firstone, '\0', MBOX_NAME_LEN+1);
      //firstone=gview->members[1];
127   strncpy(firstone,gview->members[1], MBOX_NAME_LEN);
    }
129 /* check if alone in the group */
    if(gview->nmembers==1)
131 {
      /* not yet joined? */
133   if(joined==0)
      {
135     log_info("%s is the first among the headnodes\n", myname);
        /* set parameters for pbs_server */
137     log_info("Attempting to open %s\n",svdat.watch[1].conf);
        /* set the next job id to 0 */
139     jidset(svdat.watch[1].conf, 0);
        sigsyncinit();
141     log_info("Send signal, that structure is set\n");
        notify_jbootup(getppid());
143     waitforsig();
        log_info("Waiting for bootup.. to contiue\n");
145     log_info("Got notification lets go on\n");
        //notify_jbootup(getppid());
147     log_info("done.\n");
        sigsyncunset();
149     /* joint */
        joined=1;
151   }

153 }
    /* not alone in group... */
155 else
    {
157   /* want to join ? */
      if(joined==0)
159   {
        /* set the new master */
161     strncpy(firstone,gview->members[1], MBOX_NAME_LEN);
        log_info("%s is now master\n", firstone);
163
        log_info("+ Members so far:\n");
165     for( i=1 ; i<= gview->nmembers ; i++ )
        {
167       log_info("+ %s\n", gview->members[i]);
```

```
        }
169
        /* adjust member counter */
171     members=gview->nmembers;
        log_info("--> %s is requesting the join process\n", myname);
173     /* do_join() never returns */
        do_join();
175   }
      else
177   {
        /* check if I was the last first one */
179     if(strcmp(myname,firstone)==0)
        {
181       log_info("Assisting the new member the joining process\n");
          assist_join();
183     }
        else
185     {
          log_info("I am not of any help %s %s\n",myname,firstone);
187     }

189   }


191 }
193
    /* set the new master */
195 strncpy(firstone,gview->members[1], MBOX_NAME_LEN);
    log_info("%s is now master\n", firstone);
197
    log_info("+ Members so far:\n");
199 for( i=1 ; i<= gview->nmembers ; i++ )
    {
201   log_info("+ %s\n", gview->members[i]);
    }
203
    /* adjust member counter */
205 members=gview->nmembers;
  }
207
  log_info("++++ Event ++++ handled ++++\n");
209
}
211 /*****************************************************************************
*                                                                            *
213 * handle_exec                                                              *
*                                                                            *
215 *****************************************************************************/
void handle_exec(char *msg, char *cmd, int identifier)
217 {
/*
219 * local data
*/
221
  pid_t pid;
223 char *in=NULL;
  char ** fakearg=NULL, **fakeenv=NULL;
225 char *out=NULL, *err=NULL;
  char *rsp=NULL;
227 char *sender=NULL;
  char *path=NULL;
229 int p0[2], p1[2], p2[2];
  int status;
```

```
231   /* data for select */
      fd_set fdset;
233   struct timeval tv;
      int ret;
235   int maxfd=0;

237
      /* create pipe to pump STDIN into child */
239   if(pipe(p0)!=0)
      {
241     log_err("pipe %s\n",strerror(errno));
      }
243
      /* create pipe to get STDOUT from child */
245   if(pipe(p1)!=0)
      {
247     log_err("pipe %s\n",strerror(errno));
      }
249
      /* create pipe to get STDERR from child */
251   if(pipe(p2)!=0)
      {
253     log_err("pipe %s\n",strerror(errno));
      }
255
      /* recover the STDIN from the msg */
257   in=recov_stdin(msg);

259   /* recover the argument vector from msg */
      fakearg=recov_argv(msg);
261   log_info("DEBUG: %s\n", fakearg[0]);

263   /* recover the environment vector from msg */
      fakeenv=recov_env(msg);
265
      /* recover sender from msg */
267   sender=recov_sender(msg);

269   /* recover the path from msg */
      path=recov_path(msg);
271
      /* create child process */
273   pid=fork();
      switch(pid){
275     /* error */
        case -1:  log_err("fork %s\n",strerror(errno));
277               break;
        /* child */
279     case 0:   log_info("Created child with pid %d to exec %s command\n", getpid(), cmd);
                  /* close the stdin and put the pipe on */
281               dup2(p0[0], STDIN_FILENO);

283               /* close stdout and put on the pipe */
                  dup2(p1[1], STDOUT_FILENO);
285               /* close stderr and put on the pipe */
                  dup2(p2[1], STDERR_FILENO);
287
                  /* close write for stdin pipe */
289               close(p0[1]);
                  /*close read for stdout pipe */
291               close(p1[0]);
                   /*close read for stderr pipe */
293               close(p2[0]);
```

```
                gid_t gid = recov_gid(msg);
295             if(setgid(gid)==-1)
                {
297               log_warn("setgid %s\n", strerror(errno));
                }
299             /* set stored uid and gid */
                uid_t uid = recov_uid(msg);
301             if(setuid(uid)==-1)
                {
303               log_err("setuid %s\n", strerror(errno));
                }
305             /* change cwd */
                if(chdir(path)==-1)
307             {
                  log_err("path %s\n", strerror(errno));
309             }
                /* attemt to exec command */
311             if(execve(cmd, fakearg, fakeenv)==-1)
                {
313               log_err("exec %s\n", strerror(errno));
                }
315             break;
          /* parent */
317       default: break;

319     }
        /* more parent code */
321     /* close read for stdin pipe */
        close(p0[0]);
323     /*close write for stdout pipe */
        close(p1[1]);
325     /*close write for stderr pipe */
        close(p2[1]);
327
        /* feed stdin pipe with recovered stuff */
329     if(in!=NULL)
        {
331       writefd(p0[1], in, strlen(in));
        }
333     /* close the writer side */
        close(p0[1]);
335
        /* decide which output to read */
337     /* empty fdset */
        FD_ZERO(&fdset);
339     FD_SET(p1[0], &fdset);
        FD_SET(p2[0], &fdset);
341     /* wait 5s for input */
        tv.tv_sec = 12;
343     tv.tv_usec = 0;

345     /* Calculate the greatest file descriptor in the set. */
        maxfd = p2[0];
347     if (maxfd < p1[0])
        {
349         maxfd = p1[0];
        }
351
        /* check max file escriptor */
353     ret=select(maxfd+1,&fdset,NULL,NULL,&tv);
        if(ret==-1)
355     {
          log_warn("select %s\n", strerror(errno));
```

```
357   }
      else
359   {
        if(ret>0)
361     {
          if(FD_ISSET(p1[0],&fdset))
363       {
           /* read stuff from child with builtin timeout*/
365       out=readfd(p1[0]);
          }
367       if(FD_ISSET(p2[0],&fdset))
          {
369       /* read stuff from child with builtin timeout*/
          err=readfd(p2[0]);
371       }
        }
373     else
        {
375       log_warn("select timeout\n");
        }
377   }
      close(p1[0]);
379   close(p2[0]);

381   log_info("exec returned stdout \n%s\n", out);
      log_info("exec returned stderr \n%s\n", err);
383   log_info("Return to sender %s\n", sender);

385   /* add submit and del messages to the queue, when no stderr was
       * returned */
387   if(err==NULL)
      {
389     switch(identifier)
        {
391       case(ADDMSGID):
            if(out!=NULL)
393         {
              log_info("Added job %d to internal submission queue.\n", atoi(out));
395           /* a job id is just valid once, also after a rollover */
              if(getelmt(&svdat.submitq, atoi(out))==NULL)
397           {
                addelmt(&svdat.submitq, atoi(out), msg);
399             /* increase submitted job id generation counter in case of
                 * turnover */
401             if((atoi(out)<lsub)&&(joined==1))
                {
403               /* increase turnover generation counter */
                  gensub++;
405             }
                /* set the last submitted value */
407             lsub=atoi(out);
              }
409         }
            break;
411       case(DELMSGID):
            /* only add a del message once */
413         if(getelmt(&svdat.delq, atoi(fakearg[1]))==NULL)
            {
415           log_info("Added job %d to internal deletion queue.\n", atoi(fakearg[1]));
              addelmt(&svdat.delq, atoi(fakearg[1]), msg);
417         }
            break;
419       }
```

```
421   }

423   rsp=mkrspmsg(out, err);

425   log_info("Sending..\n");
      zzz_VaSend(msgbox, CAUSAL, 0, strlen(rsp)+1 , rsp, sender, NULL);
427   log_info("done..\n");

429   /* try to get the exit status of the child */
      switch(waitpid(pid, &status, WNOHANG))
431   {
        case 0:
433       /* no child waiting, yet */
          if(kill(pid, SIGKILL)==0)
435       {
            log_warn("Child still active though I'm done, so I attemted to kill child with pid %d
                \n", pid);
437   /* give the child a chance to get killed */
      Sleep(0,1);
439       waitpid(pid, &status, WNOHANG);
          }
441       else
          {
443         log_warn("Kill failed. Zombie process with pid %d remains in process table.\n", pid);
          }
445       break;
        case -1:
447       log_warn("wait %s\n", strerror(errno));
          break;
449     default:
          /* wait successfull */
451       break;

453   }
      /* free all used memory */
455   destroystring(&rsp);
      destroystring(&err);
457   destroystring(&out);
      destroystring(&in);
459   destroylist(&fakeenv);
      destroylist(&fakearg);
461   destroystring(&sender);
      destroystring(&path);
463

465 }
    /*****************************************************************************
467 *                                                                            *
    * handle_jutex                                                               *
469 *                                                                            *
    *****************************************************************************/
471 void handle_jutex(char *msg, int identifier)
    {
473 /*
    * local data
475 */
      char *sender=NULL;
477   char *rsender=NULL;
      char *respond=NULL;
479   char *rmsg=NULL;
      int jid=0;
481
```

```
    /* recover job id */
483 jid=recov_jid(msg);

485 /* recover sender */
    sender=recov_jutex_sender(msg);
487
    switch(identifier)
489 {
      case(STRMSGID):
491     log_info("Received start message for job %d from %s\n", jid, sender);
        /* check if jid made turnover */
493     if(gensub>0)
        {
495       if(jid<ldone)
          {
497         /* adjust generation counter */
            gensub--;
499         gendone++;
          }
501     }
        /* check if jid can be executed */
503     if(checkexec(ldone, lsub, gendone, gensub, jid)==0)
        {
505       /* only allow first request to enter the job */
          if(getelmt(&svdat.jutexq, jid)==NULL)
507       {
            log_info("Executor %s has allowance to enter job %d\n", sender, jid);
509         addelmt(&svdat.jutexq, jid, msg);
            /* create start message */
511         rmsg=mkstrmsg(jid, NULL);
            /* send message to let job start */
513         zzz_VaSend(msgbox, CAUSAL, 0, strlen(rmsg)+1 , rmsg, sender, NULL);
            /* free respond and msg */
515         destroystring(&rmsg);
          }
517       else
          {
519         log_info("Executor %s was put on hold for job %d\n", sender, jid);
            addelmt(&svdat.jutexq, jid, msg);
521       }
        }
523     else
        {
525       /* release the job immediately */
    log_info("Executor %s is not allowed to enter job %d\n",sender, jid);
527       /* create finish message */
          rmsg=mkfnsmsg(jid, NULL);
529       /* send finish message */
          zzz_VaSend(msgbox, CAUSAL, 0, strlen(rmsg)+1 , rmsg, sender, NULL);
531       /* free respond and msg */
          destroystring(&rmsg);
533     }
        break;
535   case(FNSMSGID):
        log_info("Received finish message for job %d from %s\n", jid, sender);
537     /* check for generation turnover */
        if(gendone>0)
539     {
          if(jid<ldone)
541       {
            /* adjust generation counter */
543         gendone--;
          }
```

```
545         }
            /* set ldone to the current last job done */
547         if(XNOR( ldone<jid , !gendone))
            {
549           ldone=jid;
            }
551         /* release all executors by sending finish message */
            while(getelmt(&svdat.jutexq, jid)!=NULL)
553         {
              respond=getsender(&svdat.jutexq, jid);
555           rsender=recov_jutex_sender(respond);
              log_info("Realeasing executor %s\n", rsender);
557           /* create finish message */
              rmsg=mkfnsmsg(jid, NULL);
559           /* send finish message */
              zzz_VaSend(msgbox, CAUSAL, 0, strlen(rmsg)+1 , rmsg, rsender, NULL);
561           /* remove sender entry from list */
              remelmt(&svdat.jutexq, jid);
563           /* free respond and msg */
              destroystring(&rmsg);
565           destroystring(&rsender);
              //destroystring(&respond);
567         }
            /* remove the jobs done from the internal queues */
569         remelmt(&svdat.submitq, jid);
            remelmt(&svdat.delq, jid);
571

573         break;
          default:
575           log_warn("Unkown message caught from cluster\n");
      }
577   destroystring(&sender);

579 }
    /*****************************************************************************
581 *                                                                            *
    * checkexec                                                                  *
583 *                                                                            *
    *****************************************************************************/
585 int checkexec(int ldone, int lsub, int gendone, int gensub, int jid)
    {
587   /* check job sanity */
      if(XNOR(jid<=lsub, !gensub))
589   {
        log_info("LDONE: %d JID: %d LSUB: %d GENDONE: %d GENSUB: %d --> Sanity: OK\n", ldone, jid
            , lsub, gendone, gensub);
591   }
      else
593   {
        log_info("LDONE: %d JID: %d LSUB: %d GENDONE: %d GENSUB: %d --> Sanity: FALSE\n", ldone,
            jid, lsub, gendone, gensub);
595     return -1;
      }
597
      /* check first rexec */
599   if(XNOR( ldone<jid , !gendone))
      {
601     log_info("LDONE: %d JID: %d LSUB: %d GENDONE: %d GENSUB: %d --> Exec: OK\n", ldone, jid,
            lsub, gendone, gensub);
      }
603   else
      {
```

```
605     log_info("LDONE: %d JID: %d LSUB: %d GENDONE: %d GENSUB: %d -> Exec: FALSE\n", ldone,
            jid, lsub, gendone, gensub);
        return -1;
607   }

609   return 0;

611 }
/*****************************************************************************
613 *                                                                           *
* do_join                                                                   *
615 *                                                                           *
*****************************************************************************/
617 void do_join(void)
{
619   log_info("Removing normal event handler...\n");
    zzz_Remove_Upcall(msgbox);
621   zzz_Add_Upcall(msgbox_join, handle_join, USER_PRIORITY, (void *) 1);
    log_info("Join event handler initiated\n");
623   log_info("Join event handler started...\n");
    E_main_loop();
625   return;
}
627 /*****************************************************************************
*                                                                           *
629 * assist_join                                                               *
*                                                                           *
631 *****************************************************************************/
void assist_join(void)
633 {
    char *join=NULL,*finish=NULL;
635   qelmt *elmtptr;

637   /* send join information */
    join=mkjoinmsg(lsub,ldone,gensub,gendone);
639   zzz_VaSend(msgbox, SAFE, 0, strlen(join)+1 , join, "join", NULL);
    destroystring(&join);
641
    /* send all submit messages */
643   /* start at tail */
    elmtptr=svdat.submitq.tail;
645   while(elmtptr!=NULL)
    {
647     //log_info("SEND JID: %ld MSG: %s\n", elmtptr->jid, elmtptr->jmsg);
      zzz_VaSend(msgbox, CAUSAL, 0, strlen(elmtptr->jmsg)+1 , elmtptr->jmsg, "join", NULL);
649     /* move towards the head */
      elmtptr=elmtptr->prv;
651   }

653   /* send all del messages */
    /* start at tail */
655   elmtptr=svdat.delq.tail;
    while(elmtptr!=NULL)
657   {
      zzz_VaSend(msgbox, CAUSAL, 0, strlen(elmtptr->jmsg)+1 , elmtptr->jmsg, "join", NULL);
659     /* move towards the head */
      elmtptr=elmtptr->prv;
661   }

663   /* send all jmutex messages */
    /* start at tail */
665   elmtptr=svdat.jutexq.tail;
    while(elmtptr!=NULL)
```

```
667   {
        zzz_VaSend(msgbox, CAUSAL, 0, strlen(elmtptr->jmsg)+1 , elmtptr->jmsg, "join", NULL);
669     /* move towards the head */
        elmtptr=elmtptr->prv;
671   }

673   /* send finish message */
      finish=mkfnsmsg(0, NULL);
675   /* send finish message */
      log_info("Send FINISH JOIN\n");
677   zzz_VaSend(msgbox, CAUSAL, 0, strlen(finish)+1 , finish, "join", NULL);
      /* free finish msg */
679   destroystring(&finish);

681   log_info("all join data sent..\n");
      return;
683 }
    /*****************************************************************************
685 *                                                                            *
    * handle_join                                                                *
687 *                                                                            *
    *****************************************************************************/
689 void handle_join(int dummy1, void *param)
    {
691   /* message buffer for TRANSIS msgs */
      char recv_buf[MAX_MSG_SIZE];
693   int recv_type, amount;
      view *gview=NULL;
695
      amount=zzz_Receive(msgbox_join, recv_buf, MAX_MSG_SIZE, &recv_type, &gview);
697   log_info("++++ Event ++++ join event handler received message ++++ size = %d\n", amount);
      /* distinguish between gtoup chang msg and data msg */
699   /* data msg */
      if(recv_type!= VIEW_CHANGE)
701   {
        switch(recov_id(recv_buf))
703     {
          /* message received was an submit msg*/
705       case(JOIMSGID): /* recover the join msg */
                        lsub=recov_lsub(recv_buf);
707                     ldone=recov_ldone(recv_buf);
                        gensub=recov_gensub(recv_buf);
709                     gendone=recov_gendone(recv_buf);
                        log_info("JOIN: Received: LDONE: %d LSUB: %d GENDONE: %d\
711 GENSUB: %d\n", ldone, lsub, gendone, gensub);
                        log_info("Attempting to open %s\n",svdat.watch[1].conf);
713                     /* set the next job id to 0 */
                        jidset(svdat.watch[1].conf, ldone+1);
715                     sigsyncinit();
                        log_info("Send signal, that structure is set\n");
717                     notify_jbootup(getppid());
                        waitforsig();
719                     log_info("Waiting for bootup.. to contiue\n");
                        log_info("Got notification lets go on\n");
721                     log_info("done.\n");
                        sigsyncunset();
723                     /* wait for PBS to settle down */
                        Sleep(3,0);
725                     log_info("PBS TORQUE Ready\n");
                        break;
727       case(FNSMSGID): /* reinit handler */
                        log_info("Removing join handler...\n");
729                     zzz_Remove_Upcall(msgbox_join);
```

```
                        /* leave 2nd msg_box */
731                     zzz_Leave(msgbox_join, "join");
                        log_info("Reinitiate first event handler...\n");
733                     zzz_Add_Upcall(msgbox, handle_events, USER_PRIORITY, (void *) 1);
                        joined=1;
735                     log_info("--> successfully finished the join process\n");
                        log_info("++++ Event ++++ handled ++++\n");
737                     break;
        case(ADDMSGID): /* add all add msgs into internal add queue */
739                     handle_exec(recv_buf, svdat.submit, ADDMSGID);
                        break;
741     case(DELMSGID): /* add all del msgs into internal del queue */
                        handle_exec(recv_buf, svdat.del, DELMSGID);
743                     break;
        case(STRMSGID): handle_jutex(recv_buf, STRMSGID);
745                     break;
       /* message unknown and is being ignored */
747     case(UNKNOWNMSG): log_warn("Unknown message: %s\n", recv_buf);

749    }
     }
751
   return;
753 }
```

signals.h

```
   /*****************************************************************************
 2 * Project: JOSHUA                                                           *
   * Description: functions for signal handling                                *
 4 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
   *                                                                           *
 6 *           888888                    888                                   *
   *             "88b                    888                                   *
 8 *              888                    888                                   *
   *              888  .d88b.  .d8888b  88888b.  888  888  8888b.               *
10 *              888 d88""88b 88K      888 "88b 888  888     "88b              *
   *              888 888  888 "Y8888b. 888  888 888  888 .d888888              *
12 *              88P Y88..88P      X88 888  888 Y88b 888 888  888              *
   *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888              *
14 *            .d88P                                                          *
   *          .d88P"                              2006 Kai Uhlemann            *
16 *         888P"                                                             *
   *                                                                           *
18 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
   * System: Linux 2.6.8-2-686-smp on i686                                     *
20 *                                                                           *
   * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
22 *                                                                           *
   *****************************************************************************/
24 /*****************************************************************************
   *                                                                           *
26 * Headers                                                                   *
   *                                                                           *
28 *****************************************************************************/
   typedef void sigfunc(int);
30 #define notify_joshua(PID) notify( SIGUSR2, (PID))
   #define notify_jbootup(PID) notify( SIGUSR1, (PID))
32
   sigfunc *mysignal(int signr, sigfunc *sighandler);
34 void sig_usr(int signal);
   void sigbootinit(void);
```

```
36 void sigsyncinit(void);
   void notify(int signalno, pid_t pid);
38 void sigsyncunset(void);
   void waitforsig(void);
40 void sig_term(int signal);
   void sigjoshuainit(void);
42 /*****************************************************************************
   *                                                                           *
44 * timeout                                                                   *
   *                                                                           *
46 *****************************************************************************/
   /*!
48  \fn void timeout(int signal)
    \brief function react on timeouts identified by SIGALRM
50
    function provides signal handling, when timeout event occurs (SIGALRM)
52  \param signal gives integer indentifier for signal
   */
54 void timeout(int signal);
   /*****************************************************************************
56 *                                                                           *
   * shutdown                                                                  *
58 *                                                                           *
   *****************************************************************************/
60 /*!
    \fn void turndown(int signal)
62  \brief function react on shutdown request identified by SIGTERM

64  function provides signal handling, when shutdown event occurs (SIGTERM)
    \param signal gives integer indentifier for signal
66 */
   void turndown(int signal);
68 /*****************************************************************************
   *                                                                           *
70 * child                                                                     *
   *                                                                           *
72 *****************************************************************************/
   /*!
74  \fn void child(int signal)
    \brief function reacts when child process ends identified by SIGCHLD
76
    function provides signal handling, when childprocess dies (SIGCHLD)
78  \param signal gives integer indentifier for signal
   */
80 void sig_warn(int signal);
   /*****************************************************************************
82 *                                                                           *
   * spipe                                                                     *
84 *                                                                           *
   *****************************************************************************/
86 /*!
    \fn void spipe(int signal)
88  \brief function reacts when pipe breaks identified by SIGPIPE

90  function provides signal handling, when pipe breaks (SIGPIPE)
    \param signal gives integer indentifier for signal
92 */
   /* see define section for prototype */
```

signals.c

```
1 /*****************************************************************************
```

```
 * Project: JOSHUA                                                          *
 * Description: functions for signal handling                               *
 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                           *
 *                                                                          *
 *           888888                    888                                  *
 *             "88b                    888                                  *
 *              888                    888                                  *
 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
 *              888 d88""88b 88K      888 "88b 888  888     "88b             *
 *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
 *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
 *            .d88P                                                         *
 *          .d88P"                             2006 Kai Uhlemann            *
 *         888P"                                                            *
 *                                                                          *
 * Created at: Mon Nov  7 10:58:14 EST 2005                                 *
 * System: Linux 2.6.8-2-686-smp on i686                                    *
 *                                                                          *
 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.    *
 *                                                                          *
 ****************************************************************************/
/****************************************************************************
 *                                                                          *
 * Headers                                                                  *
 *                                                                          *
 ****************************************************************************/
#include "utils.h"
#include "signals.h"
#include "startup.h"
extern FILE *log_target_err;
extern FILE *log_target_out;
extern srvdata svdat;
static volatile sig_atomic_t sigflag;
static sigset_t null_mask;
/****************************************************************************
 *                                                                          *
 * mysignal                                                                 *
 *                                                                          *
 ****************************************************************************/
sigfunc *mysignal(int signr, sigfunc *sighandler)
{
  struct sigaction new_handler, old_handler;
  new_handler.sa_handler = sighandler;
  sigemptyset(&new_handler.sa_mask);
  new_handler.sa_flags = 0;
  if (signr == SIGALRM)
  {
#ifdef SA_INTERRUPT
    new_handler.sa_flags |= SA_INTERRUPT; /* Solaris */
#endif
  }
  else
  {
#ifdef SA_RESTART
    new_handler.sa_flags |= SA_RESTART; /* SVR4, BSD */
#endif
  }
  if (sigaction(signr, &new_handler, &old_handler) < 0)
  {
    return(SIG_ERR);
  }
```

```
65   return(old_handler.sa_handler);
}
67
/*************************************************************************
69 *                                                                       *
* sig_usr                                                                *
71 *                                                                       *
*************************************************************************/
73 void sig_usr(int signal)
{
75   sigflag=1;
   return;
77 }
/*************************************************************************
79 *                                                                       *
* sigbootinit                                                            *
81 *                                                                       *
*************************************************************************/
83 void sigbootinit(void)
{
85     /* set the sig_term function as signal handler for PIPE, CHLD, TERM
        * and INT */
87     if(mysignal(SIGPIPE, sig_term)==SIG_ERR)
       {
89       log_err("Unable to set signal handler for SIGPIPE.\n");
       }
91
       if(mysignal(SIGCHLD, sig_term)==SIG_ERR)
93     {
         log_err("Unable to set signal handler for SIGCHLD.\n");
95     }

97     if(mysignal(SIGTERM, sig_term)==SIG_ERR)
       {
99       log_err("Unable to set signal handler for SIGTERM.\n");
       }
101
       if(mysignal(SIGINT, sig_term)==SIG_ERR)
103    {
         log_err("Unable to set signal handler for SIGINT.\n");
105    }

107    if(mysignal(SIGHUP, sig_term)==SIG_ERR)
       {
109      log_err("Unable to set signal handler for SIGHUP.\n");
       }
111    /* init null mask */
        if(sigemptyset(&null_mask)!=0)
113    {
         log_err("Unable to set signalmask %s\n", strerror(errno));
115    }


117


119    /* no further action, just return to interupt */
       return;
121 }

123 /*************************************************************************
    *                                                                       *
125 * sigjoshuainit                                                          *
    *                                                                       *
127 *************************************************************************/
```

```
void sigjoshuainit(void)
{
    /* set the sig_term function as signal handler for PIPE, CHLD, TERM
     * and INT */
    if(mysignal(SIGPIPE, sig_warn)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGPIPE.\n");
    }

    if(mysignal(SIGTERM, sig_term)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGTERM.\n");
    }

    if(mysignal(SIGINT, sig_term)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGINT.\n");
    }

    if(mysignal(SIGHUP, sig_term)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGHUP.\n");
    }

    /* set signal handler for sigchld */
    if(mysignal(SIGCHLD, sig_warn) == SIG_ERR)
    {
        log_err("Cannot install signalhandler timeout\n");
    }

    /* init null mask */
     if(sigemptyset(&null_mask)!=0)
    {
      log_err("Unable to set signalmask %s\n", strerror(errno));
    }

    /* no further action, just return to interupt */
    return;
}
/*****************************************************************************
*                                                                            *
* sigsyncinit                                                                *
*                                                                            *
*****************************************************************************/
void sigsyncinit(void)
{
    if(mysignal(SIGUSR1, sig_usr)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGUSR1.\n");
    }

    if(mysignal(SIGUSR2, sig_usr)==SIG_ERR)
    {
      log_err("Unable to set signal handler for SIGUSE2.\n");
    }

    /* no further action, just return to interupt */
    return;
}
/*****************************************************************************
*                                                                            *
* notify                                                                     *
*                                                                            *
```

```
191 ******************************************************************************/
    void notify(int signalno, pid_t pid)
193 {
        if(kill(pid, signalno)!=0)
195     {
          log_err("kill (notify) %s\n", strerror(errno));
197     }

199
        /* no further action, just return to interupt */
201     return;
    }
203
    /*****************************************************************************
205 *                                                                            *
    * sigsyncunset                                                               *
207 *                                                                            *
    *****************************************************************************/
209 void sigsyncunset(void)
    {
211     /* set handling of USR1 and USR2 to default */
        if(mysignal(SIGUSR1, SIG_DFL)==SIG_ERR)
213     {
          log_err("Unable to reset signal handler for SIGUSR1.\n");
215     }

217     if(mysignal(SIGUSR2, SIG_DFL)==SIG_ERR)
        {
219       log_err("Unable to reset signal handler for SIGUSE2.\n");
        }
221
        /* no further action, just return to interupt */
223     return;
    }
225
    /*****************************************************************************
227 *                                                                            *
    * waitforsig                                                                 *
229 *                                                                            *
    *****************************************************************************/
231 void waitforsig(void)
    {
233     while(sigflag==0)
        {
235       sigsuspend(&null_mask);
        }
237     /* reset sflag */
        sigflag=0;
239
        /* no further action, just return to interupt */
241     return;
    }
243 /*****************************************************************************
    *                                                                            *
245 * timeout                                                                    *
    *                                                                            *
247 *****************************************************************************/
    void timeout(int signal)
249 {

251     log_warn("Operation timed out!\n");
        /* no further action, just return to interupt */
253     return;
```

```
}
/*****************************************************************************
*                                                                            *
* sig_term                                                                    *
*                                                                            *
*****************************************************************************/
void sig_term(int signal)
{
    /* GNU hack its not safe to call strsignal in signal handler */
#ifdef _GNU_SOURCE
    log_warn("%s. Shutdown initiated.\n", sys_siglist[signal]);
#else
    log_warn("%s. Shutdown initiated.\n", strsignal(signal));
#endif
//    log_warn("Shutdown initiated.\n");
    if(log_target_out!=NULL)
    {
      fclose(log_target_out);
    }
    if(log_target_err!=NULL)
    {
      fclose(log_target_err);
    }
    /* kill all processes */
    killpids(&svdat);
    /* delete internal data */
    deletedata(&svdat);
    /* exit on error */
    exit(EXIT_SUCCESS);
}
/*****************************************************************************
*                                                                            *
* shutdown                                                                   *
*                                                                            *
*****************************************************************************/
void turndown(int signal)
{
    log_warn("SIGTERM received. Shutdown initiated.\n");
    if(log_target_out!=NULL)
    {
      fclose(log_target_out);
    }
    if(log_target_err!=NULL)
    {
      fclose(log_target_err);
    }
    /* delete internal data */
    deletedata(&svdat);
    /* exit on error */
    exit(EXIT_SUCCESS);
}
/*****************************************************************************
*                                                                            *
* child                                                                      *
*                                                                            *
*****************************************************************************/
void sig_warn(int signal)
{
    /* GNU hack its not safe to call strsignal in signal handler */
#ifdef _GNU_SOURCE
    log_warn("%s. Child process got killed.\n", sys_siglist[signal]);
#else
    log_warn("%s. Child process got killed.\n", strsignal(signal));
```

```
317 #endif
        /* no further action, just return to interupt */
319     return;
    }
```

startup.h

```
 1 /*****************************************************************************
   * Project: JOSHUA                                                           *
 3 * Description: functions for startup assistence                             *
   * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
 5 *                                                                           *
   *           888888                    888                                 *
 7 *             "88b                    888                                 *
   *              888                    888                                 *
 9 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
   *              888 d88""88b 88K      888 "88b 888  888     "88b             *
11 *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
   *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
13 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
   *            .d88P                                                          *
15 *          .d88P"                              2006 Kai Uhlemann            *
   *         888P"                                                             *
17 *                                                                           *
   * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
19 * System: Linux 2.6.8-2-686-smp on i686                                     *
   *                                                                           *
21 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
   *                                                                           *
23 *****************************************************************************/
   /*****************************************************************************
25 *                                                                           *
   * Structures                                                                *
27 *                                                                           *
   *****************************************************************************/
29 /*!
     \struct progs_
31   \brief simple structure to hold name, path for exec, configuration and pid
   */
33 typedef struct progs_ {
   char *name; /* name of the program */
35 char *prgexec; /* path to exec */
   char *conf; /* path to conf */
37 pid_t pid; /* pid of program */
   } progs;
39 /*!
     \struct progs_
41   \brief structure to hold server data
   */
43 typedef struct srvdata_ {
   pid_t observer; /* pid job observer */
45 pid_t jbootup; /* pid jbootup */
   char **servers; /* head node server list */
47 progs watch[WATCHCOUNT]; /* array of processes to watch */
   char *logfile; /* logfile */
49 char *errlog; /* error logfile */
   char *submit; /* submission command */
51 char *del; /* deletion command */
   char *stat; /* status command */
53 queue submitq; /* submission queue */
   queue delq; /* deletion queue */
55 queue jutexq; /* mutex queue */
```

```
} srvdata;

/* getopt_long return codes */
enum {DUMMY_CODE=129};
/*******************************************************************************
*                                                                              *
* usage                                                                        *
*                                                                              *
*******************************************************************************/
/*!
 \fn void usage(int status)
 \brief function to printout usage information
 \param status depending on status, usage may end the program
*/
void usage (int status);
/*******************************************************************************
*                                                                              *
* decode_switches                                                              *
*                                                                              *
*******************************************************************************/
/*!
 \fn int decode_switches (int argc, char **argv);
 \brief function decode switches on startup
 \param argc argument counter
 \param **argv argument vector
*/
int decode_switches (int argc, char **argv);
/*******************************************************************************
*                                                                              *
* serverinit                                                                   *
*                                                                              *
*******************************************************************************/
/*!
 \fn void serverinit(FILE *logfile, FILE *errorlog);
 \brief function to initialize server logging facilities
 \param logfile logfile
 \param errorlog error logfile
*/
void serverinit(FILE *logfile, FILE *errorlog);
/*******************************************************************************
*                                                                              *
* bootinit                                                                     *
*                                                                              *
*******************************************************************************/
/*!
 \fn void bootinit(FILE *logfile, FILE *errorlog);
 \brief function to initialize jinit logging facilities
 \param logfile logfile
 \param errorlog error logfile
*/
void bootinit(FILE *logfile, FILE *errorlog);
/*******************************************************************************
*                                                                              *
* initdata                                                                     *
*                                                                              *
*******************************************************************************/
/*!
 \fn int initdata(srvdata *serverdata, char *configfile);
 \brief function to initialize the data structures of the server by parsing the
 configuration file
 \param *serverdata server datastructure
 \param *configfile configuration file to parse
*/
```

```
119 int initdata(srvdata *serverdata, char *configfile);
    /*****************************************************************************
121 *                                                                            *
    * deletedata                                                                 *
123 *                                                                            *
    *****************************************************************************/
125 /*!
     \fn int deletedata(srvdata *serverdata);
127  \brief function to free the data structures of the server
     \param *serverdata server datastructure
129 */
    int deletedata(srvdata *serverdata);
131 /*****************************************************************************
    *                                                                            *
133 * shutd                                                                      *
    *                                                                            *
135 *****************************************************************************/
    /*!
137  \fn void shutd(char *msg);
     \brief function to shutdown the server
139  \param *msg message for shutdown
    */
141 void shutd(char *msg);
    /*****************************************************************************
143 *                                                                            *
    * killpids                                                                   *
145 *                                                                            *
    *****************************************************************************/
147 /*!
     \fn void killpids(srvdata *serverdata);
149  \brief function to kill all pending processes on watchdog group
     \param *serverdata server datastructure
151 */
    void killpids(srvdata *serverdata);
```

startup.c

```
    /*****************************************************************************
2   * Project: JOSHUA                                                           *
    * Description: functions for startup assistence                             *
4   * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
    *                                                                           *
6   *           888888                   888                                  *
    *             "88b                   888                                  *
8   *              888                   888                                  *
    *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
10  *              888 d88""88b 88K      888 "88b 888  888     "88b             *
    *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
12  *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
    *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
14  *            .d88P                                                         *
    *          .d88P"                            2006 Kai Uhlemann            *
16  *         888P"                                                           *
    *                                                                           *
18  * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
    * System: Linux 2.6.8-2-686-smp on i686                                     *
20  *                                                                           *
    * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.     *
22  *                                                                           *
    *****************************************************************************/
24  /*****************************************************************************
    *                                                                           *
```

```
26 * Headers                                                                *
   *                                                                        *
28 *************************************************************************/
   #include "utils.h"
30 #include "startup.h"
   #include "signals.h"
32
   extern char *program_name;
34 char *configfile;
   srvdata svdat;
36
   struct option const long_options[] =
38 {
     {"help", no_argument, 0, 'h'},
40   {"version", no_argument, 0, 'V'},
     {"config", required_argument, 0, 'c'},
42   {NULL, 0, NULL, 0}
   };
44 /*************************************************************************
   *                                                                        *
46 * decode_switches                                                        *
   *                                                                        *
48 *************************************************************************/
   /* Set all the option flags according to the switches specified.
50    Return the index of the first non-option argument.  */
   int decode_switches (int argc, char **argv)
52 {
     int c;
54   configfile=NULL;

56   while ((c = getopt_long (argc, argv,
            "h"  /* help */
58          "V"  /* version */
            "c:", /* config file */
60          long_options, (int *) 0)) != EOF)
       {
62       switch (c)
   {
64   case 'V':
       printf ("joshua %s\n", VERSION);
66     exit (EXIT_SUCCESS);

68   case 'h':
       usage (EXIT_SUCCESS);
70
     case 'c':
72
             configfile=optarg;
74
       break;
76
     default:
78     usage (EXIT_FAILURE);
   }
80     }
     /* check if configfile was given */
82   if(configfile==NULL)
     {
84     fprintf(stderr, "Error: Missing config file.\n");
       usage(EXIT_FAILURE);
86   }
     return optind;
88 }
```

```
90 /*****************************************************************************
   *                                                                           *
92 * serverinit                                                                *
   *                                                                           *
94 *****************************************************************************/
   void serverinit(FILE *logfile, FILE *errorlog){
96
       /* init logging facility */
98     if(errorlog==NULL)
       {
100        set_log_target(NULL, NULL);
       }
102    else
       {
104        set_log_target(logfile, errorlog);
       }
106
       log_info("Inititiate Server startup\n");
108
       /* set signal handlers */
110    sigjoshuainit();

112    log_info("Startup finished.\n");

114    return;
   }
116
   /*****************************************************************************
118 *                                                                           *
   * bootinit                                                                *
120 *                                                                           *
   *****************************************************************************/
122 void bootinit(FILE *logfile, FILE *errorlog){

124    /* init logging facility */
       if(errorlog==NULL)
126    {
           set_log_target(NULL, NULL);
128    }
       else
130    {
           set_log_target(logfile, errorlog);
132    }

134    log_info("Inititiate Server startup\n");

136    /* set signal handlers */
       sigbootinit();
138
       log_info("Startup finished.\n");
140
       return;
142 }
   /*****************************************************************************
144 *                                                                           *
   * usage                                                                     *
146 *                                                                           *
   *****************************************************************************/
148 void usage (int status)
   {
150   printf("%s - \
   JOSHUA - JOb Scheduler for High availability Using Active/active\
```

```
152 replication\n",program_name);
    printf ("Usage: %s [OPTION]... [FILE]...\n", program_name);
154   printf ("\
Options:\n\
156   -h,        --help              display this help and exit\n\
    -V,        --version           output version information and exit\n\
158   -c config, --config config     configuration file (mandatory)\n\
");
160   exit (status);
}
162 /*****************************************************************************
*                                                                            *
164 * initdata                                                                  *
*                                                                            *
166 *****************************************************************************/
int initdata(srvdata *serverdata, char *configfile)
168 {
    cfg_opt_t opts[] =
170   {

172     CFG_STR("logfile", "/var/log/joshua/joshua.log", CFGF_NONE),
      CFG_STR("errorlog", "/var/log/joshua/joshuaerr.log", CFGF_NONE),
174     CFG_STR("scheduler_exec", "/usr/local/maui/sbin/maui", CFGF_NONE),
      CFG_STR("scheduler_conf", "/usr/local/maui/maui.cfg", CFGF_NONE),
176     CFG_STR("scheduler", "maui", CFGF_NONE),
      CFG_STR("job_server_exec", "/usr/local/sbin/pbs_server", CFGF_NONE),
178     CFG_STR("job_server_conf", "/var/spool/server_priv/serverdb", CFGF_NONE),
      CFG_STR("job_server", "pbs_server", CFGF_NONE),
180     CFG_STR("group_com_exec", "/usr/local/sbin/transis", CFGF_NONE),
      CFG_STR("group_com_conf", "/etc/transis/transis.conf", CFGF_NONE),
182     CFG_STR("group_com", "transis", CFGF_NONE),
      CFG_STR("joshua_exec", "/home/kai/joshua-0.1/joshua/joshua", CFGF_NONE),
184     CFG_STR("joshua_conf", "/etc/joshua/joshua.conf", CFGF_NONE),
      CFG_STR("joshua", "joshua", CFGF_NONE),
186     CFG_STR("submit_exec", "/usr/local/bin/qsub", CFGF_NONE),
      CFG_STR("del_exec", "/usr/local/bin/qdel", CFGF_NONE),
188     CFG_STR("stat_exec", "/usr/local/bin/qstat", CFGF_NONE),
      /* a memory leak in libconfuse forces to leave the default to NULL */
190     CFG_STR_LIST("headnodes", NULL, CFGF_NONE),
      CFG_END()
192   };
    cfg_t *cfg;
194   int i=0;
    int entries=0;
196
    cfg = cfg_init(opts, CFGF_NONE);
198
    if(cfg_parse(cfg, configfile) == CFG_PARSE_ERROR)
200   {
      /* no output will be seen when uninitialized */
202     log_err("parsing configfile\n");
    }
204
    serverdata->jbootup=getpid();
206   /* parse trough config data and store into internal serverdata */
    /* TODO cpystr */
208   serverdata->watch[0].name=cpystr(cfg_getstr(cfg, "scheduler"));
    serverdata->watch[0].prgexec=cpystr(cfg_getstr(cfg, "scheduler_exec"));
210   serverdata->watch[0].conf=cpystr(cfg_getstr(cfg, "scheduler_conf"));
    serverdata->watch[0].pid=0;
212   serverdata->watch[1].name=cpystr(cfg_getstr(cfg, "job_server"));
    serverdata->watch[1].prgexec=cpystr(cfg_getstr(cfg, "job_server_exec"));
214   serverdata->watch[1].conf=cpystr(cfg_getstr(cfg, "job_server_conf"));
```

```
    serverdata->watch[1].pid=0;
216 serverdata->watch[2].name=cpystr(cfg_getstr(cfg, "group_com"));
    serverdata->watch[2].prgexec=cpystr(cfg_getstr(cfg, "group_com_exec"));
218 serverdata->watch[2].conf=cpystr(cfg_getstr(cfg, "group_com_conf"));
    serverdata->watch[2].pid=0;
220 serverdata->watch[3].name=cpystr(cfg_getstr(cfg, "joshua"));
    serverdata->watch[3].prgexec=cpystr(cfg_getstr(cfg, "joshua_exec"));
222 serverdata->watch[3].conf=cpystr(cfg_getstr(cfg, "joshua_conf"));
    serverdata->watch[3].pid=0;
224
    /* logging */
226 serverdata->logfile=cpystr(cfg_getstr(cfg, "logfile"));
    serverdata->errlog=cpystr(cfg_getstr(cfg, "errorlog"));
228
    /* cmdtools */
230 serverdata->submit=cpystr(cfg_getstr(cfg, "submit_exec"));
    serverdata->del=cpystr(cfg_getstr(cfg, "del_exec"));
232 serverdata->stat=cpystr(cfg_getstr(cfg, "stat_exec"));

234 /* alloc server list */
    entries=cfg_size(cfg, "headnodes");
236 serverdata->servers=(char **)malloc(sizeof(char*)*(entries+1));
    if(serverdata->servers==NULL)
238 {
      log_err("malloc %s\n",strerror(errno));
240 }

242 /* servers */
    for(i=0; i<entries; i++)
244 {
      serverdata->servers[i]=cpystr(cfg_getnstr(cfg, "headnodes" , i));
246 }
    serverdata->servers[i]=NULL;
248 cfg_free(cfg);

250 /* init queues */
    initq(&serverdata->submitq);
252 initq(&serverdata->delq);
    initq(&serverdata->jutexq);
254
    return 0;
256 }

258 int deletedata(srvdata *serverdata)
{
260   int i=0;

262   for(i=0; i<WATCHCOUNT; i++)
    {
264     destroystring(&serverdata->watch[i].name);
      destroystring(&serverdata->watch[i].prgexec);
266   destroystring(&serverdata->watch[i].conf);
    }
268
    /* logging */
270 destroystring(&serverdata->logfile);
    destroystring(&serverdata->errlog);
272
    /* cmdtools */
274 destroystring(&serverdata->submit);
    destroystring(&serverdata->del);
276 destroystring(&serverdata->stat);
```

```
278   /* servers */
      destroylist(&serverdata->servers);
280
      /* free queues */
282   destroyq(&serverdata->submitq);
      destroyq(&serverdata->delq);
284   destroyq(&serverdata->jutexq);

286
      return 0;
288
}
290 /*****************************************************************************
*                                                                            *
292 * shutd                                                                      *
*                                                                            *
294 *****************************************************************************/
void shutd(char *msg){
296
if(msg!=NULL)
298 {
  log_warn("%s\n",msg);
300 }
/* close log files */
302 raise(SIGTERM);

304 }
/*****************************************************************************
306 *                                                                            *
* killpids                                                                   *
308 *                                                                            *
*****************************************************************************/
310 void killpids(srvdata *serverdata)
{
312   int i=0;
    /* kill everything which has a pid with SIGTERM */
314   for(i=0; i<WATCHCOUNT; i++)
    {
316     if(serverdata->watch[i].pid!=0)
      {
318       kill(serverdata->watch[i].pid, SIGTERM);
      }
320   }
    /* kill everything which has a pid again with SIGKILL*/
322   for(i=0; i<WATCHCOUNT; i++)
    {
324     if(serverdata->watch[i].pid!=0)
      {
326       kill(serverdata->watch[i].pid, SIGKILL);
      }
328   }
}
```

A.3.4 libjutils

data.h

```
1 /*****************************************************************************
* Project: JOSHUA                                                            *
```

```
3 * Description: data definitions and structures for the JOSHUA components       *
  * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                                *
5 *                                                                              *
  *           888888                    888                                     *
7 *             "88b                    888                                     *
  *              888                    888                                     *
9 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.                 *
  *              888 d88""88b 88K      888 "88b 888  888     "88b               *
11 *             888 888  888 "Y8888b. 888  888 888  888 .d888888               *
  *              88P Y88..88P      X88 888  888 Y88b 888 888  888               *
13 *             888  "Y88P"   88888P' 888  888  "Y88888 "Y888888               *
  *           .d88P                                                            *
15 *          .d88P"                           2006 Kai Uhlemann               *
  *         888P"                                                              *
17 *                                                                            *
  * Created at: Mon Nov  7 10:58:14 EST 2005                                     *
19 * System: Linux 2.6.8-2-686-smp on i686                                       *
  *                                                                            *
21 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
  *                                                                            *
23 ******************************************************************************/
  /*!
25   \file data.h
     \brief contains all data structures useful for server or client
27
     Here all structs, enums and defines of any kind used for server
29   or client will be stored and may so easy changes or extended.
  */
31 /******************************************************************************
  *                                                                            *
33 * HTML Main Page Costumization                                               *
  *                                                                            *
35 ******************************************************************************/
  /*! \mainpage JOSHUA Documentation
37  *
   * \section intro_sec Introduction
39  *
   * This is the introduction.
41  *
   * \section install_sec Installation
43  *
   * \subsection step1 Step 1: Opening the box
45  *
   * etc...
47 */
  /******************************************************************************
49 *                                                                            *
  * Structures                                                                 *
51 *                                                                            *
  ******************************************************************************/
53 /*!
     \struct server_qs
55   \brief simple structure to read the first byted out of the pbs serverdb

57   This is a structure which facilitates parts of the pbs_server serverdb.
     The structure has been extracted from the pbs code in torque 2.0p1
59 */

61 struct server_qs {
       int  sv_numjobs;  /* number of job owned by server    */
63     int  sv_numque;   /* nuber of queues managed          */
       int  sv_jobidnumber;  /* next number to use in new jobid  */
65          /* the server struct must be saved  */
```

```
            /* whenever this value is updated   */
67      time_t sv_savetm; /* time of server db update         */
} sv_qs;
69
#define XNOR(A,B) (((A)&&(B))||((!(A))&&(!(B))))
71
/*!
73  \def WATCHCOUNT
    set number of processes to watch to 4
75 */
#define WATCHCOUNT 4
77
/*****************************************************************************
79 *                                                                           *
   * Macros                                                                    *
81 *                                                                           *
   *****************************************************************************/
83


85
/*!
87  \def HEADNODEGROUP
    set string for the headnodegroup to "headmasters"
89 */
#define HEADNODEGROUP "headmasters"
91
/*!
93  \def STARTSIZE
    set startsize for allocation to 8
95 */
#define STARTSIZE 8
97
/*!
99  \def REGFILE
    set value for regular file to 1
101 */
#define REGFILE 1
103
/*!
105  \def NREGFILE
    set non regular file to 0
107 */
#define NREGFILE 0
109
/*!
111  \def ENTLEN
    set regular entry length of a msg string to 5
113  dont change that till sollutiion for sprintf is found
   */
115 #define ENTLEN 4

117 /*!
     \def PRTLEN
119  set regular printf format string of a msg string to "%5d"
     dont change that till sollutiion for sprintf is found
121 */
#define PRTLEN "%4x"
123
/*!
125  \def UIDLEN
    set UID length in a msg to 5
127  dont change that till sollutiion for sprintf is found
   */
```

```
129 #define UIDLEN 4

131 /*!
     \def PRTUID
133  set regular printf format string of a uid string to "%5d"
     dont change that till sollutiion for sprintf is found
135 */
    #define PRTUID "%4x"
137
    /*!
139  \def GIDLEN
     set GID length in a msg to 5
141  dont change that till sollutiion for sprintf is found
    */
143 #define GIDLEN 4

145 /*!
     \def PRTGID
147  set regular printf format string of a uid string to "%5d"
     dont change that till sollutiion for sprintf is found
149 */
    #define PRTGID "%4x"
151
    /*!
153  \def MSGHLEN
     set length for a message header to 3
155  dont change that either
    */
157 #define MSGHLEN 3

159 /*!
     \def ADDMSG
161  set submit/add message identifier to "add"
    */
163 #define ADDMSG "add"

165 /*!
     \def RSPMSG
167  set response message identifier to "rsp"
    */
169 #define RSPMSG "rsp"

171 /*!
     \def RSPMSG
173  set status message identifier to "sta"
    */
175 #define STAMSG "sta"

177 /*!
     \def DELMSG
179  set deletion message identifier to "del"
    */
181 #define DELMSG "del"

183 /*!
     \def STRMSG
185  set start message identifier to "str"
    */
187 #define STRMSG "str"

189 /*!
     \def FNSMSG
191  set finish message identifier to "fns"
```

```
*/
#define FNSMSG "fns"

/*!
 \def JOIMSG
 set joi message identifier to "joi"
*/
#define JOIMSG "joi"


/*!
 \def ADDMSGID
 set submit/add message identifier to 1
*/
#define ADDMSGID 1

/*!
 \def RSPMSGID
 set response message identifier to 2
*/
#define RSPMSGID 2

/*!
 \def STAMSGID
 set status message identifier to 3
*/
#define STAMSGID 3

/*!
 \def DELMSGID
 set del message identifier to 4
*/
#define DELMSGID 4

/*!
 \def STRMSGID
 set start message identifier to 5
*/
#define STRMSGID 5

/*!
 \def FNSMSGID
 set finished message identifier to 6
*/
#define FNSMSGID 6

/*!
 \def JOIMSGID
 set join message identifier to 7
*/
#define JOIMSGID 7


/*!
 \def UNKNOWNMSG
 set submit/add message identifier to 0
*/
#define UNKNOWNMSG 0

/*!
 \def LOG_INFO
 set logging information identifier to 1
```

```
255 */
#define LOG_INFO 1
257
/*!
259  \def LOG_WARN
 set logging warning identifier to 2
261 */
#define LOG_WARN 2
263
/*!
265  \def LOG_ERR
 set logging information identifier to 3
267 */
#define LOG_ERR 3
269
/*!
271  \def IDUID
 set uid message identifier to 1
273 */
#define IDUID 1
275
/*!
277  \def IDGID
 set gid message identifier to 2
279 */
#define IDGID 2
281

283 /*!
 \def IDJID
285  set job message identifier to 3
*/
287 #define IDJID 3

289 /*!
 \def IDARGV
291  set job message identifier to 4
*/
293 #define IDARGV 4

295 /*!
 \def IDARGV
297  set job message identifier to 5
*/
299 #define IDENV 5

301 /*!
 \def IDLSU
303  set last submitted job message identifier to 6
*/
305 #define IDLSU 6

307 /*!
 \def IDLDO
309  set last job done message identifier to 7
*/
311 #define IDLDO 7

313 /*!
 \def IDGSU
315  set generation submitted job message identifier to 8
*/
317 #define IDGSU 8
```

```
319 /*!
     \def IDGDO
321  set generation job done message identifier to 9
    */
323 #define IDGDO 9

325 /*!
     \def PBS_SEQNUMTOP
327  set maximum PBS job id value to 99999999
    */
329 #define PBS_SEQNUMTOP 99999999

331
    /*!
333  \def EXIT_PBS_SUCCESS
     set exit value for PBS job success to 0
335 */
    #define EXIT_PBS_SUCCESS 0
337

339 /*!
     \def EXIT_PBS_ABORT
341  set exit value for PBS job abortion to 1
    */
343 #define EXIT_PBS_ABORT 1
```

list.h

```
1  /*****************************************************************************
   * Project: JOSHUA                                                           *
3  * Description: double linked queue q                                        *
   * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
5  *                                                                           *
   *           888888                   888                                  *
7  *             "88b                   888                                  *
   *              888                   888                                  *
9  *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
   *              888 d88""88b 88K      888 "88b 888  888     "88b             *
11 *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
   *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
13 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
   *            .d88P                                                          *
15 *          .d88P"                             2006 Kai Uhlemann            *
   *         888P"                                                             *
17 *                                                                           *
   * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
19 * System: Linux 2.6.8-2-686-smp on i686                                    *
   *                                                                           *
21 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.    *
   *                                                                           *
23 *****************************************************************************/
   /*****************************************************************************
25 *                                                                           *
   * Macro functions                                                           *
27 *                                                                           *
   *****************************************************************************/
29 #define qsize(q) ((q)->size)
   #define qhead(q) ((q)->head)
31 #define qtail(q) ((q)->tail)
   #define addelmt(QUEUE, JID, MSG) inselmt(QUEUE, createlmt(JID, MSG))
33 #define getmsg(QUEUE, JID) getelmt(QUEUE, JID)->jmsg
```

```
#define getsender(QUEUE, JID) getelmt(QUEUE, JID)->jmsg
/*****************************************************************************
*                                                                            *
*  structure for linked list elements                                        *
*                                                                            *
*****************************************************************************/
/*!
  \struct qelmt_
  \brief double linked list to manage the internal jobqueue

  This is a structure which facilitates the jobmessages from the clients, hold
  in this internal queue list
*/
typedef struct qelmt_ {
unsigned long jid; /* job identifier */
char *jmsg; /* job message */
struct qelmt_ *next; /* pointer to next element */
struct qelmt_ *prv; /* pointer to previous element */
} qelmt;
/*****************************************************************************
*                                                                            *
*  structure for linked list                                                 *
*                                                                            *
*****************************************************************************/
/*!
  \struct queue_
  \brief manage the internal jobqueue
  \param size size of queue

  This is a structure holds the internal job queue
*/
typedef struct queue_ {
unsigned int size; /* size of the queue */
qelmt *head; /* head element of the queue */
qelmt *tail; /* tail element of the queue */
} queue;
/*****************************************************************************
*                                                                            *
*  prototypes                                                                *
*                                                                            *
*****************************************************************************/
/*****************************************************************************
*                                                                            *
* initq                                                                      *
*                                                                            *
*****************************************************************************/
/*!
 \fn void initq(queue *q)
 \brief function to initialize the internal jobqueue

 function initializes the internal jobqueue given by argument, it set the size to
 0 and head and tail of the queue to NULL
 \param *q jobqueue
*/
void initq(queue *q);
/*****************************************************************************
*                                                                            *
* destroyq                                                                   *
*                                                                            *
*****************************************************************************/
/*!
 \fn void destroyq(queue *q)
 \brief function to free the internal jobqueue
```

```
97
 function frees the internal jobqueue given by argument, it frees all elements of
99  the queue
 \param *q jobqueue
101 */
void destroyq(queue *q);
103 /*****************************************************************************
*                                                                            *
105 * createlmt                                                                 *
*                                                                            *
107 *****************************************************************************/
/*!
109  \fn qelmt *createlmt(unsigned long jid, char *msg)
 \brief function to create queue elements
111
 function is used to create a queue element out of a jobid and the jobmessage
113  createlmt returns a queue element or NULL in case of failure
 \param jid the jobidentifier
115  \param jmsg the jobmessage
 \return queue element or NULL in case of failure
117 */
qelmt *createlmt(unsigned long jid, char *msg);
119 /*****************************************************************************
*                                                                            *
121 * destroyelmt                                                               *
*                                                                            *
123 *****************************************************************************/
/*!
125  \fn int destroyelmt(qelmt **element)
 \brief function to free queue elements
127
 function is used to free a queue element
129  detroyelmt returns 0 or -1 in case of failure
 \param element element to free
131  \return 0 or -1 in case of failure
*/
133 int destroyelmt(qelmt **element);
/*****************************************************************************
135 *                                                                            *
* inselmt                                                                   *
137 *                                                                            *
*****************************************************************************/
139 /*!
 \fn int inselmt(queue *q, qelmt *element)
141  \brief function to insert queue elements into the internal queue

143  function is inserts the given element into the given queue. Since jobqueue works
 like a FIFO, the job is always added at the head of the queue.
145  inselmt returns 0 or -1 in case of failure
 \param *q jobqueue
147  \param element element to add
 \return 0 or -1 in case of failure
149 */
int inselmt(queue *q, qelmt *element);
151 /*****************************************************************************
*                                                                            *
153 * getelmt                                                                   *
*                                                                            *
155 *****************************************************************************/
/*!
157  \fn qelmt *getelmt(queue *q, unsigned long jid)
 \brief function to get queue elements
159
```

```
 function is used to get a queue element out of a jobid a
 getelmt returns a queue element or NULL in case of failure
 \param *q jobqueue
 \param jid the jobidentifier
 \return queue element or NULL in case of failure
*/
qelmt *getelmt(queue *q, unsigned long jid);
/*****************************************************************************
*                                                                            *
* remelmt                                                                    *
*                                                                            *
*****************************************************************************/
/*!
 \fn int remelmt(queue *q, unsigned long jid)
 \brief function to remove queue element from the internal queue

 function removes an element by given job identifier from the given queue.
 remelmt returns 0 or -1 in case of failure
 \param *q jobqueue
 \param element element to add
 \return 0 or -1 in case of failure
*/
int remelmt(queue *q, unsigned long jid);
```

list.c

```
/*****************************************************************************
* Project: JOSHUA                                                            *
* Description: double linked queue q                                         *
* Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
*                                                                            *
*           888888                    888                                  *
*             "88b                    888                                  *
*              888                    888                                  *
*              888  .d88b.  .d8888b   88888b.  888  888  8888b.             *
*              888 d88""88b 88K       888 "88b 888  888     "88b            *
*              888 888  888 "Y8888b.  888  888 888  888 .d888888            *
*              88P Y88..88P      X88  888  888 Y88b 888 888  888            *
*              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888            *
*            .d88P                                                           *
*          .d88P"                              2006 Kai Uhlemann             *
*         888P"                                                              *
*                                                                            *
* Created at: Mon Nov  7 10:58:14 EST 2005                                   *
* System: Linux 2.6.8-2-686-smp on i686                                      *
*                                                                            *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
*                                                                            *
*****************************************************************************/
/*****************************************************************************
*                                                                            *
* Headers                                                                    *
*                                                                            *
*****************************************************************************/
#include "utils.h"
/*****************************************************************************
*                                                                            *
* initq                                                                      *
*                                                                            *
*****************************************************************************/
void initq(queue *q)
{
```

```
q->size = 0;
q->head = NULL;
q->tail = NULL;
return;
}
/*****************************************************************************
*                                                                            *
* destroyq                                                                   *
*                                                                            *
*****************************************************************************/
void destroyq(queue *q) {

qelmt *elmtptr, *tmpptr;

/* start at head to destroy queue */
elmtptr=q->head;

/* free until end of queue is reached */
while (elmtptr!=NULL)
{
  /* save ptr position */
  tmpptr=elmtptr;
  /* go to next element */
  elmtptr=elmtptr->next;
  /* free the element */
  destroyelmt(&tmpptr);
}
/* even destroy all data in queue structure */
memset(q, 0, sizeof(queue));

return;

}
/*****************************************************************************
*                                                                            *
* createelmt                                                                 *
*                                                                            *
*****************************************************************************/
qelmt *createlmt(unsigned long jid, char *msg)
{

qelmt *element;

/* test msg */
if(msg==NULL)
{
  log_warn("q message was empty\n");
  return NULL;
}
/* initialize the new element */
/* allocate storage for the element */
element = (qelmt *)malloc(sizeof(qelmt));
if(element==NULL)
{
  log_err("malloc %s\n",strerror(errno));
}
/* start allocating for the message */
element->jmsg = (char *)malloc(sizeof(char)*(strlen(msg)+1));
if(element->jmsg==NULL)
{
  log_err("malloc %s\n",strerror(errno));
}
/* set memory */
```

```
100 memset(element->jmsg, '\0', strlen(msg)+1);
    /* copy the q job message */
102 strncpy(element->jmsg,msg,strlen(msg));
    /* set the rest */
104 element->jid = jid;
    element->next=NULL;
106 element->prv=NULL;

108 return element;

110 }
    /*****************************************************************************
112 *                                                                            *
    * destroyelmt                                                                *
114 *                                                                            *
    *****************************************************************************/
116 int destroyelmt(qelmt **element){

118   if(*element==NULL)
      {
120     return EXIT_FAILURE;
      }
122   /* destroy the job message string first */
      destroystring(&(*element)->jmsg);
124   /* then free the element */
      free(*element);
126   *element=NULL;
      return EXIT_SUCCESS;
128
    }
130 /*****************************************************************************
    *                                                                            *
132 * inselmt                                                                    *
    *                                                                            *
134 *****************************************************************************/
    int inselmt(queue *q, qelmt *element){
136
    qelmt *elmtptr;
138
    /* Insert the element into the q */
140 /* find correct position in q */
    /* if first element */
142 if(q->head==NULL)
    {
144   q->head=element;
      q->tail=element;
146   //log_info("Inserted JOB %ld to internal list\n", element->jid);
      return 0;
148 }
    /* if not insert at the head of the queue */
150 elmtptr=q->head;
    /* set backlink of old head to new element */
152 elmtptr->prv=element;
    /* set the forward link of the new element to the old head */
154 element->next=elmtptr;
    /* new element is now new head */
156 q->head=element;
    //log_info("Inserted JOB %ld to internal list\n", element->jid);
158 return 0;

160 }
    /*****************************************************************************
162 *                                                                            *
```

```
* getelmt                                                                    *
164 *                                                                         *
*****************************************************************************/
166 qelmt *getelmt(queue *q, unsigned long jid){

168   unsigned long tailjid=0, headjid=0;
      qelmt *elmtptr=NULL;
170
      if(q->head==NULL){
172     /* nothing yet */
        return NULL;
174   }

176   /* get the head and tail jids */
      tailjid=q->tail->jid;
178   headjid=q->head->jid;

180   /* which has shortest distance */
      if(labs(tailjid-jid)>labs(headjid-jid))
182   {
        /* start at tail */
184     elmtptr=q->tail;
        /* search until head of queue is reached */
186     while (elmtptr!=NULL)
        {
188       /* check jid */
          if(elmtptr->jid==jid)
190       {
            return elmtptr;
192       }
          /* go to next element */
194       elmtptr=elmtptr->prv;
        }
196

198   }
      else
200   {
        /* start at head */
202     elmtptr=q->head;
        /* search until tail of queue is reached */
204     while (elmtptr!=NULL)
        {
206       /* check jid */
          if(elmtptr->jid==jid)
208       {
            return elmtptr;
210       }
          /* go to next element */
212       elmtptr=elmtptr->next;
        }
214
      }
216
    return NULL;
218
    }
220
    /*****************************************************************************
222 *                                                                           *
    * remelmt                                                                   *
224 *                                                                           *
    *****************************************************************************/
```

```
226 int remelmt(queue *q, unsigned long jid){

228   qelmt *elmtptr=NULL, *tmpptr=NULL, *next=NULL, *prev=NULL;

230   /* get the element to remove */
      elmtptr=getelmt(q, jid);
232   if(elmtptr==NULL)
      {
234     return -1;
      }
236   /* if element to remove is head */
      if(elmtptr==q->head)
238   {
        /* save ptr position */
240     tmpptr=elmtptr;
        /* go to next element */
242     elmtptr=elmtptr->next;
        /* check whether there is a next element */
244     if(elmtptr!=NULL)
        {
246       /* set new head */
          q->head=elmtptr;
248       elmtptr->prv=NULL;
        }
250     /* reset the head and tail pointer since noone is left */
        else
252     {
          q->head=NULL;
254       q->tail=NULL;
        }
256     //log_info("Removed JOB %ld from internal list at head\n", tmpptr->jid);
        /* free the element */
258     destroyelmt(&tmpptr);
        return 0;
260   }
       /* if element to remove is tail */
262   if(elmtptr==q->tail)
      {
264     /* save ptr position */
        tmpptr=elmtptr;
266     /* go to previous element */
        elmtptr=elmtptr->prv;
268     /* check whether there is a previous element */
        if(elmtptr!=NULL)
270     {
          /* set new tail */
272       q->tail=elmtptr;
          elmtptr->next=NULL;
274     }
        /* reset the tail and head pointer, since there is no element anymore */
276     else
        {
278       q->head=NULL;
          q->tail=NULL;
280     }
        //log_info("Removed JOB %ld from internal list at tail\n", tmpptr->jid);
282     /* free the element */
        destroyelmt(&tmpptr);
284     return 0;
      }
286   /* if somewhere in between */
      /* save ptr position */
288   tmpptr=elmtptr;
```

```
  /* go to next element */
290 next=elmtptr->next;
  /* go to previous element */
292 prev=elmtptr->prv;
  /* set next and prv pointers */
294 next->prv=prev;
  prev->next=next;
296 //log_info("Removed JOB %ld from internal list in between\n", tmpptr->jid);
  /* free the element */
298 destroyelmt(&tmpptr);
  return 0;
300
}
```


log.h

```
1 /*****************************************************************************
  * Project: JOSHUA                                                           *
3 * Description: logging related function for all JOSHUA components           *
  * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                            *
5 *                                                                           *
  *           888888                    888                                 *
7 *             "88b                    888                                 *
  *              888                    888                                 *
9 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.              *
  *              888 d88""88b 88K      888 "88b 888  888     "88b             *
11 *              888 888  888 "Y8888b. 888  888 888  888 .d888888             *
  *              88P Y88..88P      X88 888  888 Y88b 888 888  888             *
13 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888             *
  *            .d88P                                                          *
15 *          .d88P"                             2006 Kai Uhlemann           *
  *         888P"                                                             *
17 *                                                                           *
  * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
19 * System: Linux 2.6.8-2-686-smp on i686                                    *
  *                                                                           *
21 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.    *
  *                                                                           *
23 *****************************************************************************/
  /*****************************************************************************
25 *                                                                           *
  * Macro functions                                                           *
27 *                                                                           *
  *****************************************************************************/
29 #define log_info(FORMAT, args...) log_all(LOG_INFO, (FORMAT), ## args)
  #define log_warn(FORMAT, args...) log_all(LOG_WARN, (FORMAT), ## args)
31 #define log_err(FORMAT, args...) log_all(LOG_ERR, (FORMAT), ## args)
  /*****************************************************************************
33 *                                                                           *
  * Function prototypes                                                       *
35 *                                                                           *
  *****************************************************************************/
37 /*****************************************************************************
  *                                                                           *
39 * get logdate                                                               *
  *                                                                           *
41 *****************************************************************************/
  /*!
43  \fn char *getlogdate()
   \brief function to get the system time
45
   function to get the local time as a string from the system
```

```
47  getlogdate returns a string or NULL in case  of a failure
    \return a string or NULL in case  of a failure
49 */
   char *getlogdate();
51 /*****************************************************************************
   *                                                                           *
53 * set_log_target                                                            *
   *                                                                           *
55 *****************************************************************************/
   /*!
57  \fn void set_log_target(FILE *out, FILE *err)
    \brief function to set the target file streams for the logging facilty
59
    function set the target to the given filestreams. If the parameters are NULL
61  stdout and stderr are used
    \param *out the normal output target for information and warnings
63  \param *err the error output target for errors
   */
65 void set_log_target(FILE *out, FILE *err);
   /*****************************************************************************
67 *                                                                           *
   * log                                                                       *
69 *                                                                           *
   *****************************************************************************/
71 /*!
    \fn void log_all(int loglevel, const char *format, ...);
73  \brief function to log events

75  function logs events to the file streams given by set_log_target, depending on
    the loglevel. The function is somewhat generic and is used to provide log_err,
77  log_warn and log_info
    \param loglevel distingiushes between information, errors and warnings
79  \param *format format string for message, similar to printf() format string
   */
81 void log_all(int loglevel, const char *format, ...);
   /*****************************************************************************
83 *                                                                           *
   * log_info                                                                  *
85 *                                                                           *
   *****************************************************************************/
87 /*!
    \fn void log_info(const char *format, ...);
89  \brief function to log information events

91  function logs information events to the file streams given by set_log_target.
    \param *format format string for message, similar to printf() format string
93 */
   /* see define section for prototype */
95 /*****************************************************************************
   *                                                                           *
97 * log_err                                                                   *
   *                                                                           *
99 *****************************************************************************/
   /*!
101 \fn void log_err(const char *format, ...);
    \brief function to log error events
103
    function logs error events to the file streams given by set_log_target
105 and raises SIGTERM
    \param *format format string for message, similar to printf() format string
107 */
   /* see define section for prototype */
109 /*****************************************************************************
```

```
*                                                                            *
* log_warn                                                                   *
*                                                                            *
******************************************************************************/
/*!
 \fn void log_warn(const char *format, ...);
 \brief function to log information events

 function logs warning events to the file streams given by set_log_target.
 \param *format format string for message, similar to printf() format string
*/
/* see define section for prototype */
```

log.c

```
/******************************************************************************
 * Project: JOSHUA                                                            *
 * Description: logging related function for all JOSHUA components            *
 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
 *                                                                            *
 *           888888                    888                                    *
 *             "88b                    888                                    *
 *              888                    888                                    *
 *              888  .d88b.  .d8888b  88888b.  888  888  8888b.               *
 *              888 d88""88b 88K      888 "88b 888  888     "88b              *
 *              888 888  888 "Y8888b. 888  888 888  888 .d888888              *
 *              88P Y88..88P      X88 888  888 Y88b 888 888  888              *
 *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888              *
 *            .d88P                                                           *
 *          .d88P"                              2006 Kai Uhlemann             *
 *         888P"                                                              *
 *                                                                            *
 * Created at: Mon Nov  7 10:58:14 EST 2005                                   *
 * System: Linux 2.6.8-2-686-smp on i686                                      *
 *                                                                            *
 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
 *                                                                            *
 ******************************************************************************/
/******************************************************************************
 *                                                                            *
 * Headers                                                                    *
 *                                                                            *
 ******************************************************************************/
#include "utils.h"
/******************************************************************************
 *                                                                            *
 * Global data                                                                *
 *                                                                            *
 ******************************************************************************/
char *program_name =NULL;
FILE *log_target_err = NULL;
FILE *log_target_out = NULL;
/******************************************************************************
 *                                                                            *
 * get logdate                                                                *
 * return  - allocated string with logging date and time on success or NULL   *
 *                                                                            *
 ******************************************************************************/
char *getlogdate()
{
  /*
  * local data
```

```
   */
49 time_t curtime;
   struct tm *timest;
51
   curtime = time(NULL);
53 timest = localtime(&curtime);

55 return asctime(timest);

57 }
/*****************************************************************************
59 *                                                                           *
* set_log_target                                                             *
61 * FILE *out - set the log output for information to out                     *
* FILE *err - set the log output for errors and warnings to err              *
63 *                                                                           *
*****************************************************************************/
65 void set_log_target(FILE *out, FILE *err)
{
67
   if(out!=NULL)
69 {
     log_target_out = out;
71 }
   else
73 {
     log_target_out = stdout;
75 }

77 if(err!=NULL)
   {
79   log_target_err = err;
   }
81 else
   {
83   log_target_err = stderr;
   }
85 }
/*****************************************************************************
87 *                                                                           *
* log_all                                                                    *
89 * int loglevel - distiquish between information , warning and error level   *
* cont char *format - format string for log message                          *
91 *                                                                           *
*****************************************************************************/
93 void log_all(int loglevel, const char* format, ...)
{
95 /*
   * local data
97 */
   char *timebuf=NULL;
99 va_list vp;

101 /* return immediatly if no format is given */
   if(format == NULL)
103 {
     return;
105 }
   /* return immediatly if log targets not set */
107 if((log_target_out == NULL)||(log_target_err==NULL))
   {
109   return;
   }
```

```
111
    /* get the time */
113 timebuf = getlogdate();
    timebuf[24] = '\0';
115
    /* start parameter list */
117 va_start(vp, format);

119 /* print message depending on log level */
    switch(loglevel)
121 {
      /* just an info message */
123 case LOG_INFO:
      fprintf(log_target_out, "[%s] %s =%d= Info: ", timebuf, program_name, getpid());
125   vfprintf(log_target_out, format, vp);
      fflush(log_target_out);
127   break;
      /* a warning */
129 case LOG_WARN:
      fprintf(log_target_err, "[%s] %s =%d= Warning: ", timebuf, program_name, getpid());
131   vfprintf(log_target_err, format, vp);
      fflush(log_target_err);
133   break;
      /* error message */
135 case LOG_ERR:
      fprintf(log_target_err, "[%s] %s =%d= Error: ", timebuf, program_name, getpid());
137   vfprintf(log_target_err, format, vp);
      fflush(log_target_err);
139   /* when an error occurs, shutdown */
      raise(SIGTERM);
141   break;
    default:
143   break;
    }
145 /* end parameter sweep */
    va_end(vp);
147

149 }
```

msg.h

```
1 /*************************************************************************
  * Project: JOSHUA                                                       *
3 * Description: message related functions for joshua and jcmd            *
  * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                        *
5 *                                                                       *
  *           888888                    888                             *
7 *             "88b                    888                             *
  *              888                    888                             *
9 *              888  .d88b.  .d8888b   88888b.  888  888  8888b.         *
  *              888 d88""88b 88K       888 "88b 888  888     "88b       *
11 *             888 888  888 "Y8888b. 888  888 888  888 .d888888        *
  *              88P Y88..88P      X88 888  888 Y88b 888 888  888        *
13 *             888  "Y88P"   88888P' 888  888  "Y88888 "Y888888        *
  *           .d88P                                                      *
15 *         .d88P"                           2006 Kai Uhlemann          *
  *        888P"                                                        *
17 *                                                                     *
  * Created at: Mon Nov  7 10:58:14 EST 2005                            *
19 * System: Linux 2.6.8-2-686-smp on i686                              *
  *                                                                     *
```

```
21 * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
   *                                                                            *
23 ******************************************************************************/
   /******************************************************************************
25 *                                                                            *
   * Macro functions                                                            *
27 *                                                                            *
   ******************************************************************************/
29 #define recov_str_header(MSG) recov_msgentry((MSG), 1)
   #define recov_str_uid(MSG) recov_msgentry((MSG), 2)
31 #define recov_str_jid(MSG) recov_msgentry((MSG), 2)
   #define recov_str_lsub(MSG) recov_msgentry((MSG), 2)
33 #define recov_str_gid(MSG) recov_msgentry((MSG), 3)
   #define recov_str_ldone(MSG) recov_msgentry((MSG), 3)
35 #define recov_str_stdin(MSG) recov_msgentry((MSG), 4)
   #define recov_str_gensub(MSG) recov_msgentry((MSG), 4)
37 #define recov_str_argv(MSG) recov_msgentry((MSG), 5)
   #define recov_str_gendone(MSG) recov_msgentry((MSG), 5)
39 #define recov_str_env(MSG) recov_msgentry((MSG), 6)
   #define recov_str_sender(MSG) recov_msgentry((MSG), 7)
41 #define recov_uid(MSG) recov_ids((MSG), IDUID)
   #define recov_gid(MSG) recov_ids((MSG), IDGID)
43 #define recov_jid(MSG) recov_ids((MSG), IDJID)
   #define recov_lsub(MSG) recov_ids((MSG), IDLSU)
45 #define recov_ldone(MSG) recov_ids((MSG), IDLDO)
   #define recov_gensub(MSG) recov_ids((MSG), IDGSU)
47 #define recov_gendone(MSG) recov_ids((MSG), IDGDO)
   #define recov_stdin(MSG) recov_msgentry((MSG), 4)
49 #define recov_sender(MSG) recov_msgentry((MSG), 7)
   #define recov_path(MSG) recov_msgentry((MSG), 8)
51 #define recov_argv(MSG) recov_vector((MSG), IDARGV)
   #define recov_env(MSG) recov_vector((MSG), IDENV)
53 #define recov_stdout(MSG) recov_msgentry((MSG), 2)
   #define recov_stderr(MSG) recov_msgentry((MSG), 3)
55 #define mkaddmsg(A1, A2, A3, A4, A5, A6, A7) mkcmdmsg(A1, A2, A3, A4, A5, A6, A7, ADDMSGID)
   #define mkdelmsg(A1, A2, A3, A4, A5, A6, A7) mkcmdmsg(A1, A2, A3, A4, A5, A6, A7, DELMSGID)
57 #define mkstamsg(A1, A2, A3, A4, A5, A6, A7) mkcmdmsg(A1, A2, A3, A4, A5, A6, A7, STAMSGID)
   #define mkstrmsg(JID, SENDER) mkjutexmsg(JID, SENDER, STRMSGID)
59 #define mkfnsmsg(JID, SENDER) mkjutexmsg(JID, SENDER, FNSMSGID)
   #define recov_jutex_sender(MSG) recov_msgentry((MSG), 3)
61 /******************************************************************************
   *                                                                            *
63 * Function prototypes                                                        *
   *                                                                            *
65 ******************************************************************************/
   /******************************************************************************
67 *                                                                            *
   * packlist                                                                   *
69 *                                                                            *
   ******************************************************************************/
71 /*!
    \fn char *packlist(char **listvec)
73  \brief function to pack list of strings, like argv into a dynamic string

75  function to pack a list of strings, such as the argument vector
    and the environment into a buffer to make jcmds transparent to
77  remotely called functions
    packall returns a fully allocated string with flatened list vector
79  \param **listvec a vector like argv or env to be packed in string
    \return allocated string with flatened list vector
81 */
   char *packlist(char **listvec);
83 /******************************************************************************
```

```
*                                                                    *
* rebuildvector                                                      *
*                                                                    *
**********************************************************************/
/*!
 \fn char **rebuildlist(char *flatlist)
 \brief function to uppack the a string to get argv and env

 function to upack pack everything out of a string to get the argument vector
 and the environment to make jcmds transparent to remotely called functions
 packall returns a fully allocated string list
 \param *flatlist a flatened list packed in string
 \return allocated exploded list vector
*/
char **rebuildvector(char *flaitlist);
/**********************************************************************
*                                                                    *
* mkmsg                                                              *
*                                                                    *
**********************************************************************/
/*!
 \fn char* mkaddmsg(char *msg, char *append)
 \brief generic function to creates messages for TRANSIS

 function creates a message in appending the append string to the msg including
 the length information header
 mkaddmsg returns a fully allocated string
 \param *msg the msg so far
 \param *appen the string to be appended to the message
 \return a fully allocated message string
*/
char *mkmsg(char **msg, char *append);
/**********************************************************************
*                                                                    *
* mkaddmsg                                                           *
*                                                                    *
**********************************************************************/
/*!
 \fn char* mkaddmsg(uid_t uid, gid_t gid, char *in, char **argvec,\
  char **envvec, char *sender, char *path)
 \brief function creates add/submit message for TRANSIS

 function creates a message for submitting a job via jsub to the HEADMASTER
 group. The included data will be used to invoke a qsub there
 mkaddmsg returns a fully allocated string
 \param uid the user id
 \param gid the group id
 \param *in the standard input stored as string
 \param **argvec the argument vector
 \param **envvec the environment vector
 \param *sender the message sender
 \param *path the working directory
 \return a fully allocated message string
*/
char* mkcmdmsg(uid_t uid, gid_t gid, char *in, char **argvec, char **envvec,
    char *sender, char *path, int identifier);
/**********************************************************************
*                                                                    *
* recov_msgentry                                                     *
*                                                                    *
**********************************************************************/
/*!
 \fn char *recov_msgentry(char *msg, int entryno)
```

```
147  \brief function recovers any entry given by entrynumber from TRANSIS message

149  function recover the content of a message (a the string, using the entry number
     this function is somewhat generic and is used for recovery
151  recov_msgentry returns a fully allocated string or NULL in case of failure
     \param *msg the message as string
153  \param entryno number of entry to recover
     \return a msg or UNKNOWNMSG in case of a failure
155 */
    char *recov_msgentry(char *msg, int entryno);
157 /*****************************************************************************
    *                                                                           *
159 * recov_id                                                                  *
    *                                                                           *
161 *****************************************************************************/
    /*!
163  \fn int recov_id(char *msg)
     \brief function recovers msgid from TRANSIS message
165
     function recover the identifier of a message (not the string but an
167  integer identifier), using the identifier approporiate action can be taken
     recov_msgid returns a msg id defined in data.h or UNKNOWNMSG in case
169  of a failure
     \param *msg the message as string
171  \return a msgi ID or UNKNOWNMSG in case of a failure
    */
173 int recov_id(char *msg);
    /*****************************************************************************
175 *                                                                           *
    * recov_ids                                                                 *
177 *                                                                           *
    *****************************************************************************/
179 /*!
     \fn int recov_ids(char *msg, int identifier)
181  \brief function recovers ids by identifier from TRANSIS message

183  function recover the ids of a message (as an integer), using the id number
     this function is somewhat generic and is used for recov_uid, recov_gid and
185  recov_jid
     recov_msgentry returns an int or -1 in case of failure
187  \param *msg the message as string
     \param identifier number of id to recover
189  \return an integer id or -1 in case of a failure
    */
191 int recov_ids(char *msg, int identifier);
    /*****************************************************************************
193 *                                                                           *
    * recov_vector                                                              *
195 *                                                                           *
    *****************************************************************************/
197 /*!
     \fn char **recov_vector(char *msg, int identifier)
199  \brief function recovers any entry given by entrynumber from TRANSIS message

201  function recover the content of a message (as a vector), using the identifier
     this function is somewhat generic and is used for recovery of argv and env
203  recov_vector returns a fully allocated vector or NULL in case of failure
     \param *msg the message as string
205  \param identifier number of entry to recover
     \return a vector or NULL in case of a failure
207 */
    char **recov_vector(char *msg, int identifier);
209 /*****************************************************************************
```

```
*                                                                            *
* recov_uid                                                                  *
*                                                                            *
******************************************************************************/
/*!
 \fn int recov_uid(char *msg)
 \brief function recovers user id from TRANSIS message

 function recover the user identifier of a message (not the string but an
 uid_t identifier), using the identifier approporiate action can be taken
 recov_uid returns a uid_t on success or -1 in case  of a failure
 \param *msg the message as string
 \return the user id in msg or -1 in case of failure
*/
/* see define section for prototype */
/*****************************************************************************
*                                                                            *
* recov_gid                                                                  *
*                                                                            *
******************************************************************************/
//! function recovers group id from TRANSIS message
/*!
 \fn int recov_gid(char *msg)
 \brief function recovers group id from TRANSIS message

 function recover the group identifier of a message (not the string but an
 uid_t identifier), using the identifier approporiate action can be taken
 recov_uid returns a gid_t on success or -1 in case of a failure
 \param *msg the message as string
 \return the group in msg or -1 in case of failure
*/
/* see define section for prototype */
/*****************************************************************************
*                                                                            *
* recov_jid                                                                  *
*                                                                            *
******************************************************************************/
//! function recovers job id from TRANSIS message
/*!
 \fn int recov_jid(char *msg)
 \brief function recovers group id from TRANSIS message

 function recover the job identifier of a message (not the string but an
 integer identifier), using the identifier approporiate action can be taken
 recov_uid returns an int on success or -1 in case of a failure
 \param *msg the message as string
 \return the group in msg or -1 in case of failure
*/
/* see define section for prototype */
/*****************************************************************************
*                                                                            *
* recov_stdin                                                                *
*                                                                            *
******************************************************************************/
/*!
 \fn char *recov_stdin(char *msg)
 \brief function recovers stdin from TRANSIS message

 function recovers the standard input from a message as a string, using
 it, the stdin approporiate action can be taken
 recov_stdin returns a string or NULL in case of a failure or empty stdin
 \param *msg the message as string
 \return fully allocated string containing the STDIN of NULL when failure
```

```
273 */
    /* see define section for prototype */
275 /*****************************************************************************
    *                                                                           *
277 * recov_argv                                                                *
    *                                                                           *
279 *****************************************************************************/
    /*!
281  \fn char **recov_argv(char *msg)
     \brief function recovers the argument vector from TRANSIS message
283
     function recovers the argument vector from a message as a string list, using
285  it, the argv approporiate action can be taken
     recov_argv returns a string list or NULL in case of a failure
287  \param *msg the message as string
     \return the argument vector in a string list or NULL in case of a failure
289 */
    /* see define section for prototype */
291 /*****************************************************************************
    *                                                                           *
293 * recov_env                                                                 *
    *                                                                           *
295 *****************************************************************************/
    /*!
297  \fn char **recov_env(char *msg);
     \brief function recovers the enironment from TRANSIS message
299
     function recovers the environment from a message as a string list, using
301  the environment approporiate action can be taken
     recov_env returns a string list or NULL in case of a failure
303  \param *msg the message as string
     \return the environment vector in a string list or NULL in case of a failure
305 */
    /* see define section for prototype */
307 /*****************************************************************************
    *                                                                           *
309 * recov_sender                                                              *
    *                                                                           *
311 *****************************************************************************/
    /*!
313  \fn char **recov_sender(char *msg);
     \brief function recovers the sender from TRANSIS message
315
     function recovers the sender from a message as a string, using
317  recov_sender returns a string or NULL in case of a failure
     \param *msg the message as string
319  \return the sender in a string list or NULL in case of a failure
    */
321 /* see define section for prototype */
    /*****************************************************************************
323 *                                                                           *
    * mkrspmsg                                                                  *
325 *                                                                           *
    *****************************************************************************/
327 /*!
     \fn char *mkrspmsg(char *out, char *err);
329  \brief function creates response messgage to deliver stdout/stderr to client

331  function creates response messgage to redirect stdout/stderr to client needed
     for transparent gateway like execution of client commands
333  mkrspmsg returns a fully allocated vector or NULL in case of failure
     \param *out stdout string
335  \param *err stderr string
```

```
 \return a string or NULL in case of a failure
*/
char *mkrspmsg(char *out, char *err);
/*****************************************************************************
*                                                                            *
* mkjutexmsg                                                                 *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *mkjutexmsg(int jid, char *sender, int identifier);
 \brief function creates mutex messgage

 function creates mutex messgage to realize cluster mutual exclusion
 mkjutexmsg returns a fully allocated vector or NULL in case of failure
 \param jid job identifier
 \param *sender sender of message
 \param identifier message identifier
 \return a string or NULL in case of a failure
*/
char *mkjutexmsg(int jid, char *sender, int identifier);
/*****************************************************************************
*                                                                            *
* mkjoinmsg                                                                  *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *mkjoinmsg(int lsub, int ldone, int gensub, int gendone);
 \brief function creates join messgage

 function creates join messgage to send important server values
 mkjoinmsg returns a fully allocated vector or NULL in case of failure
 \param ldone last job done
 \param lsub last job submitted
 \param gendone generation counter for jobs done
 \param gensub generation counter for jobs submitted
 \return a string or NULL in case of a failure
*/
char *mkjoinmsg(int lsub, int ldone, int gensub, int gendone);
```

msg.c

```
/*****************************************************************************
* Project: JOSHUA                                                            *
* Description: message related functions for joshua and jcmd                 *
* Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
*                                                                            *
*           888888                    888                                  *
*             "88b                    888                                  *
*              888                    888                                  *
*              888  .d88b.  .d8888b   88888b.  888  888  8888b.              *
*              888 d88""88b 88K       888 "88b 888  888     "88b             *
*              888 888  888 "Y8888b.  888  888 888  888 .d888888             *
*              88P Y88..88P      X88  888  888 Y88b 888 888  888             *
*              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888             *
*            .d88P                                                           *
*          .d88P"                              2006 Kai Uhlemann             *
*         888P"                                                              *
*                                                                            *
* Created at: Mon Nov  7 10:58:14 EST 2005                                   *
* System: Linux 2.6.8-2-686-smp on i686                                      *
*                                                                            *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
```

```
22 *                                                                            *
   ******************************************************************************/
24 /******************************************************************************
   *                                                                            *
26 * Headers                                                                    *
   *                                                                            *
28 ******************************************************************************/
   #include "utils.h"
30 /******************************************************************************
   *                                                                            *
32 * packlist                                                                   *
   *                                                                            *
34 ******************************************************************************/
   char *packlist(char **listvec)
36 {
     /*
38   * local data
     */
40   int entries=0, length=0;
     char *flatlist=NULL, *entry=NULL, entlen[ENTLEN+1];
42
     /* set the memory of entlen */
44   memset(entlen, '\0', ENTLEN+1);

46   /* count all entries and calculate length */
     if(listvec!=NULL){
48     while(listvec[entries]!=NULL)
       {
50       length+=strlen(listvec[entries++]);
       }
52   }
     /* for every entry additional ENTLEN bytes are needed */
54   length+=entries*ENTLEN;

56   /* start allocating */
     flatlist = (char *)malloc(sizeof(char)*length+1);
58   if(flatlist==NULL)
     {
60     log_err("malloc %s\n",strerror(errno));
     }
62   /* start generating the flatened list */
     entries=0;
64
     /* initialize string */
66   memset(flatlist, '\0', length+1);

68   /* fill the flatlist with items */
     if(listvec!=NULL)
70   {
       while((entry=listvec[entries])!=NULL)
72     {
         sprintf(entlen,PRTLEN,strlen(listvec[entries]));
74       strncat(flatlist,entlen, ENTLEN);
         strncat(flatlist,entry,strlen(listvec[entries++]));
76     }
     }
78   return flatlist;
   }
80 /******************************************************************************
   *                                                                            *
82 * rebuildvector                                                              *
   *                                                                            *
84 ******************************************************************************/
```

```
char **rebuildvector(char *flatlist)
{
  /*
  * local data
  */
  int entries=0,length=0;
  char *listptr=flatlist, entlen[ENTLEN+1], *entity=NULL;
  char **rebuild=NULL;

  /* intit used char fields */
  memset(entlen, '\0', ENTLEN+1);

  /* if flatlist empty return NULL */
  if(flatlist==NULL)
  {
    return NULL;
  }

  /* go through the string to get the number of entries*/
  while(strlen(listptr)>0)
  {
    strncpy(entlen,listptr,ENTLEN);
    /* get the length of the next entry and check if it could be fetched */
    if(sscanf(entlen,PRTLEN,&length)==0)
    {
      log_warn("Vector rebuilt failed.\n");
      return NULL;
    }
    entries++;
    /* jump to next entry */
    listptr+=length+ENTLEN;
  }

  /* allocate space for the new list */
  rebuild = (char **)malloc(sizeof(char*)*(entries+1));
  if(rebuild==NULL)
  {
    log_err("malloc %s\n",strerror(errno));
  }

  /* go through the string again a put the strings into the list */
  listptr=flatlist;
  entries=0;

  while(strlen(listptr)>0)
  {
    strncpy(entlen,listptr,ENTLEN);
    /* get the length of the next entry and check if it could be fetched */
    if(sscanf(entlen,PRTLEN,&length)==0)
    {
      log_warn("Vector rebuilt failed.\n");
      return NULL;
    }

    /* start allocating */
    entity = (char *)malloc(sizeof(char)*(length+1));
    if(entity==NULL)
    {
      log_err("malloc %s\n",strerror(errno));
    }

    /* intit used char fields */
    memset(entity, '\0', length+1);
```

```
148
      /* jump to string entry */
150   listptr+=ENTLEN;

152   /* copy the list entity */
      strncpy(entity,listptr,length);
154   /* put the entity into the list */
      rebuild[entries++]=entity;
156   /* jump to next length entry */
      listptr+=length;
158
    }
160 /* last entity is set to NULL */
    rebuild[entries]=NULL;
162 return rebuild;

164 }

166 /*****************************************************************************
*                                                                            *
168 * mkmsg                                                                      *
*                                                                            *
170 *****************************************************************************/
    char *mkmsg(char **msg, char *append)
172 {
    /*
174 * local data
    */
176 int length=0;
    char *strptr=*msg;
178 char entlen[ENTLEN+1];
    char *newmsg=NULL;
180
    /* set the memory of entlen */
182 memset(entlen, '\0', ENTLEN+1);

184 /* check sanity of append */
    if(append!=NULL)
186 {
      /* get string length */
188   length+=strlen(append);
      /* print length into entlen field */
190   sprintf(entlen, PRTLEN, strlen(append));
      length+=ENTLEN;
192
    }
194 /* if append was not sane/empty */
    else
196 {
      /* add zero length field return the msg */
198   /* print length into entlen field */
      sprintf(entlen, PRTLEN, 0);
200   length+=ENTLEN;
    }
202
    /* if this is an empty msg */
204 if(strptr==NULL)
    {
206
      /* if append was empty */
208   if(append==NULL)
      {
210     return strptr;
```

```
    }
212 /* else create a new msg */
    else
214 {
     /* alloc the space for the string */
216 newmsg = (char *)malloc(sizeof(char)*length+1);
    if(newmsg==NULL)
218 {
      log_err("malloc %s\n",strerror(errno));
220 }
    /* empty memory */
222 memset(newmsg, '\0', length+1);

224 /* add the length field */
    strncat(newmsg, entlen, ENTLEN);
226 /* append content */
    strncat(newmsg, append, strlen(append));
228
    /* reset msg pointer */
230 *msg=newmsg;

232 /* return newmsg */
    return newmsg;
234
    }
236
  }
238 /* if there has been alread some msg add the append */
  else
240 {
     /* get newmsg length */
242 length+=strlen(strptr);

244 /* if append was empty */
    if(append==NULL)
246 {
      /* return the msg with an empty length field appended */
248   /* realloc for the the new msg size */
      newmsg=(char *)realloc(strptr, length+1);
250   if(newmsg==NULL)
      {
252     log_err("malloc %s\n",strerror(errno));
      }
254   /* set pointers */
      strptr=newmsg;
256
      /* append the empty length field */
258   strncat(strptr, entlen, ENTLEN);

260   /* make sure the new msg is terminated */
      strptr[length]='\0';
262
      /* reset msg pointer */
264   *msg=strptr;

266   /* return strptr */
      return strptr;
268 }
    /* append msgfield to existing msg */
270 else
    {
272   /* return the msg with an length field appended and msgfield */
      /* realloc for the the new msg size */
```

```
274         newmsg=(char *)realloc(strptr, length+1);
            if(newmsg==NULL)
276         {
              log_err("malloc %s\n",strerror(errno));
278         }
            /* set pointers */
280         strptr=newmsg;

282         /* append the length field */
            strncat(strptr, entlen, ENTLEN);
284
            /* append content */
286         strncat(strptr, append, strlen(append));

288         /* make sure the new msg is terminated */
            strptr[length]='\0';
290
            /* reset msg pointer */
292         *msg=strptr;

294         /* return strptr */
            return strptr;
296
        }
298
    }
300 return NULL;
}
302 /*****************************************************************************
*                                                                            *
304 * mkcmdmsg                                                                   *
*                                                                            *
306 *****************************************************************************/
char *mkcmdmsg(uid_t uid, gid_t gid, char *in, char **argvec, char **envvec,
308     char *sender, char *path, int identifier)
{
310 /*
* local data
312 */
  char *uidstr=NULL, *gidstr=NULL, *argstr=NULL, *envstr=NULL, *msg=NULL;
314

316 /* get the string size for the uid and allocate space */
  if((uidstr=(char*)malloc((snprintf(NULL, 0, "%d",
318 (int)uid)+1)*sizeof(char)))==NULL)
  {
320   fprintf(stderr, "malloc %s\n", strerror(errno));
    exit(EXIT_FAILURE);
322 }

324 /* create the uidstr */
  snprintf(uidstr, snprintf(NULL, 0, "%d", (int)uid)+1, "%d",
326 (int)uid);

328 /* get the string size for the gid and allocate space */
  if((gidstr=(char*)malloc((snprintf(NULL, 0, "%d",
330 (int)gid)+1)*sizeof(char)))==NULL)
  {
332   fprintf(stderr, "malloc %s\n", strerror(errno));
    exit(EXIT_FAILURE);
334 }

336 /* create the gidstr */
```

```
    snprintf(gidstr, snprintf(NULL, 0, "%d", (int)gid)+1, "%d",
338 (int)gid);

340   /* create argument string */
    argstr=packlist(argvec);
342
    /* create environment string */
344   envstr=packlist(envvec);

346   /* append all the strings to an add/submit message */
    switch(identifier)
348   {
      case(ADDMSGID):
350     msg=mkmsg(&msg,ADDMSG);
        break;
352   case(STAMSGID):
        msg=mkmsg(&msg,STAMSG);
354     break;
      case(DELMSGID):
356     msg=mkmsg(&msg,DELMSG);
        break;
358   default:
       fprintf(stderr, "Request to built unknown message format. Exiting.\n");
360    exit(EXIT_FAILURE);
       break;
362 }

364 msg=mkmsg(&msg,uidstr);
    msg=mkmsg(&msg,gidstr);
366 msg=mkmsg(&msg,in);
    msg=mkmsg(&msg,argstr);
368 msg=mkmsg(&msg,envstr);
    msg=mkmsg(&msg,sender);
370 msg=mkmsg(&msg,path);

372 /* free temporary buffers */
    destroystring(&uidstr);
374 destroystring(&gidstr);
    destroystring(&argstr);
376 destroystring(&envstr);

378 return msg;

380 }

382 /*****************************************************************************
*                                                                            *
384 * recov_msgentry                                                           *
*                                                                            *
386 *****************************************************************************/
char *recov_msgentry(char *msg, int entryno)
388 {
    /*
390 * local data
    */
392 char entlen[ENTLEN+1];
    char *entry=NULL;
394 int length=0, i=0;
    char *strptr=msg;
396
    /* set the memory of entlen */
398 memset(entlen, '\0', ENTLEN+1);
```

```
400   /* if there is no msg, i call it unknown */
      if(strptr==NULL)
402   {
        log_warn("Message recognition failed.\n");
404     return NULL;
      }
406
      /* is entryno sane */
408   if(entryno<=0)
      {
410     log_warn("Message recognition failed.\n");
        return NULL;
412   }

414   for(i=0; i<entryno; i++)
      {
416     /* get the next entry length */
        if(strlen(strptr)>=ENTLEN)
418     {
          strncpy(entlen, strptr, ENTLEN);
420     }
        else
422     {
          log_warn("Message recognition failed.\n");
424       return NULL;
        }
426
        /* get the length of the next entry and check if it could be fetched */
428     if(sscanf(entlen,PRTLEN,&length)==0)
        {
430       log_warn("Message recognition failed.\n");
          return NULL;
432     }
        /* move strptr forward to next length entry */
434     strptr+=length+ENTLEN;
      }
436     /* jump back to where the entry starts */
        strptr-=length;
438
        /* check if any entry was in msg, if not return NULL */
440     if(length<=0)
        {
442       return NULL;
        }
444
        /* if there was anything allocate the space for the string */
446     entry = (char *)malloc(sizeof(char)*(length+1));
        if(entry==NULL)
448     {
          log_err("malloc %s\n",strerror(errno));
450     }

452     /* intit used char fields */
        memset(entry, '\0', length+1);
454
        if(strlen(strptr)>=length)
456     {
          strncpy(entry, strptr, length);
458     }
        else
460     {
          log_warn("Message recognition failed.\n");
462       destroystring(&entry);
```

```
      return NULL;
464   }

466 return entry;

468 }

470
/*****************************************************************************
472 *                                                                           *
* recov_id                                                                  *
474 *                                                                           *
*****************************************************************************/
476 int recov_id(char *msg)
{
478   /*
  * local data
480   */
  char *msgid=NULL;
482
  /* recover only if msg is sane */
484   if(msg==NULL)
  {
486     log_warn("Message could not be recognized\n");
    return UNKNOWNMSG;
488   }

490   /* get the id string */
  msgid=recov_str_header(msg);
492
  /* work only further when string is sane */
494   if(msgid==NULL)
  {
496     log_warn("Message could not be recognized\n");
    destroystring(&msgid);
498     return UNKNOWNMSG;
  }
500

502   /* check which msg arrived */
  if(strncmp(msgid, ADDMSG, strlen(msgid))==0)
504   {
    destroystring(&msgid);
506     return ADDMSGID;
  }
508
  /* check which msg arrived */
510   if(strncmp(msgid, DELMSG, strlen(msgid))==0)
  {
512     destroystring(&msgid);
    return DELMSGID;
514   }

516   /* check which msg arrived */
  if(strncmp(msgid, STAMSG, strlen(msgid))==0)
518   {
    destroystring(&msgid);
520     return STAMSGID;
  }
522
  /* check which msg arrived */
524   if(strncmp(msgid, STRMSG, strlen(msgid))==0)
  {
```

```
526     destroystring(&msgid);
        return STRMSGID;
528   }
      /* check which msg arrived */
530   if(strncmp(msgid, FNSMSG, strlen(msgid))==0)
      {
532     destroystring(&msgid);
        return FNSMSGID;
534   }

536   /* check which msg arrived */
      if(strncmp(msgid, RSPMSG, strlen(msgid))==0)
538   {
        destroystring(&msgid);
540     return RSPMSGID;
      }
542
      /* check which msg arrived */
544   if(strncmp(msgid, JOIMSG, strlen(msgid))==0)
      {
546     destroystring(&msgid);
        return JOIMSGID;
548   }

550   destroystring(&msgid);
      return UNKNOWNMSG;
552 }
    /*****************************************************************************
554 *                                                                           *
    * recov_ids                                                                 *
556 *                                                                           *
    *****************************************************************************/
558 int recov_ids(char *msg, int identifier)
    {
560   /*
      * local data
562   */
      char *msgid=NULL;
564   int id=0;

566   /* recover only if msg is sane */
      if(msg==NULL)
568   {
        log_warn("Message could not be recognized\n");
570     return -1;
      }
572
      /* recover appropriate string */
574   switch(identifier)
      {
576     case IDUID:
          msgid=recov_str_uid(msg);
578       break;
        case IDGID:
580       msgid=recov_str_gid(msg);
          break;
582     case IDJID:
          msgid=recov_str_jid(msg);
584       break;
        case IDLSU:
586       msgid=recov_str_lsub(msg);
          break;
588     case IDLDO:
```

```
      msgid=recov_str_ldone(msg);
590     break;
    case IDGSU:
592     msgid=recov_str_gensub(msg);
      break;
594   case IDGDO:
      msgid=recov_str_gendone(msg);
596     break;
    default:
598     return -1;
  }
600
  /* was the recered string sane? */
602 if(msgid==NULL)
  {
604   log_warn("Message could not be recognized\n");
    destroystring(&msgid);
606   return -1;
  }
608

610 /* check which msg arrived */
  if(sscanf(msgid, "%d", &id)==1)
612 {

614   destroystring(&msgid);
    return id;
616 }

618 log_warn("Message could not be recognized\n");
  destroystring(&msgid);
620 return -1;
}
622 /*****************************************************************************
*                                                                            *
624 * recov_vector                                                               *
*                                                                            *
626 *****************************************************************************/
char **recov_vector(char *msg, int identifier)
628 {
  /*
630 * local data
  */
632 char *list=NULL;
  char **vector=NULL;
634
  /* recover when msg was sane */
636 if(msg==NULL)
  {
638   log_warn("Message could not be recognized\n");
    return NULL;
640 }

642 /* recover appropriate list */
  switch(identifier)
644 {
    case IDARGV:
646     list=recov_str_argv(msg);
      break;
648   case IDENV:
      list=recov_str_env(msg);
650     break;
    default:
```

```
652         return NULL;
    }
654   /* check whether list is sane */
    if(list==NULL)
656   {
      log_warn("Message could not be recognized\n");
658     destroystring(&list);
      return NULL;
660   }

662   /* create vector from list */
    vector=rebuildvector(list);
664
    if(vector!=NULL)
666   {

668     destroystring(&list);
      return vector;
670   }

672   log_warn("Message could not be recognized\n");
    destroystring(&list);
674   return NULL;
  }
676 /*****************************************************************************
  *                                                                           *
678 * mkrspmsg                                                                  *
  *                                                                           *
680 *****************************************************************************/
  char *mkrspmsg(char *out, char *err)
682 {
  /*
684 * local data
  */
686   char *msg=NULL;

688
    /* append all the strings to an add/submit message */
690   msg=mkmsg(&msg,RSPMSG);
    msg=mkmsg(&msg,out);
692   msg=mkmsg(&msg,err);

694   return msg;

696 }
  /*****************************************************************************
698 *                                                                           *
  * mkjutexmsg                                                                *
700 *                                                                           *
  *****************************************************************************/
702 char *mkjutexmsg(int jid, char *sender, int identifier)
  {
704 /*
  * local data
706 */
    char *msg=NULL;
708   char *jidstr=NULL;

710   /* get the string size for the jid and allocate space */
    if((jidstr=(char*)malloc((snprintf(NULL, 0, "%d",
712 (int)jid)+1)*sizeof(char)))==NULL)
    {
714   fprintf(stderr, "malloc %s\n", strerror(errno));
```

```
    }
716
    /* create the jidstr */
718   snprintf(jidstr, snprintf(NULL, 0, "%d", (int)jid)+1, "%d",
(int)jid);
720
    /* append all the strings to an start/finish message */
722   switch(identifier)
    {
724     case(STRMSGID):
          msg=mkmsg(&msg,STRMSG);
726       break;
        case(FNSMSGID):
728       msg=mkmsg(&msg,FNSMSG);
          break;
730     default:
         fprintf(stderr, "Request to built unknown message format. Exiting.\n");
732      exit(EXIT_FAILURE);
        break;
734   }

736   msg=mkmsg(&msg,jidstr);
    msg=mkmsg(&msg,sender);
738
    /* free temporary buffers */
740   destroystring(&jidstr);

742   return msg;

744 }

746
/*****************************************************************************
748 *                                                                           *
* mkjoinmsg                                                                 *
750 *                                                                           *
*****************************************************************************/
752 char *mkjoinmsg(int lsub, int ldone, int gensub, int gendone)
{
754 /*
* local data
756 */
    char *msg=NULL;
758   char *lsubstr=NULL;
    char *ldonestr=NULL;
760   char *gensubstr=NULL;
    char *gendonestr=NULL;
762
    /* get the string size for lsub and allocate space */
764   if((lsubstr=(char*)malloc((snprintf(NULL, 0, "%d",
(int)lsub)+1)*sizeof(char)))==NULL)
766   {
      log_err("malloc %s\n", strerror(errno));
768   }

770   /* create the lsubstr */
    snprintf(lsubstr, snprintf(NULL, 0, "%d", (int)lsub)+1, "%d",
772 (int)lsub);

774   /* get the string size for ldone and allocate space */
    if((ldonestr=(char*)malloc((snprintf(NULL, 0, "%d",
776 (int)ldone)+1)*sizeof(char)))==NULL)
    {
```

```
778     log_err("malloc %s\n", strerror(errno));
    }
780
    /* create the ldonestr */
782   snprintf(ldonestr, snprintf(NULL, 0, "%d", (int)ldone)+1, "%d",
(int)ldone);
784
    /* get the string size for gensub and allocate space */
786   if((gensubstr=(char*)malloc((snprintf(NULL, 0, "%d",
(int)gensub)+1)*sizeof(char)))==NULL)
788   {
      log_err("malloc %s\n", strerror(errno));
790   }

792   /* create the gensubstr */
      snprintf(gensubstr, snprintf(NULL, 0, "%d", (int)gensub)+1, "%d",
794 (int)gensub);

796   /* get the string size for gendone and allocate space */
      if((gendonestr=(char*)malloc((snprintf(NULL, 0, "%d",
798 (int)gendone)+1)*sizeof(char)))==NULL)
      {
800     log_err("malloc %s\n", strerror(errno));
      }
802
      /* create the gendonestr */
804   snprintf(gendonestr, snprintf(NULL, 0, "%d", (int)gendone)+1, "%d",
(int)gendone);
806
      /* append all the strings to an start/finish message */
808   msg=mkmsg(&msg,JOIMSG);
      msg=mkmsg(&msg,lsubstr);
810   msg=mkmsg(&msg,ldonestr);
      msg=mkmsg(&msg,gensubstr);
812   msg=mkmsg(&msg,gendonestr);

814   /* free temporary buffers */
      destroystring(&lsubstr);
816   destroystring(&ldonestr);
      destroystring(&gensubstr);
818   destroystring(&gendonestr);


820
      return msg;
822
}
```

utils.h

```
/*************************************************************************
2 * Project: JOSHUA                                                      *
  * Description: utilitiy functions for all JOSHUA components            *
4 * Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                       *
  *                                                                       *
6 *           888888                  888                               *
  *             "88b                  888                               *
8 *              888                  888                               *
  *              888  .d88b.  .d8888b  88888b.  888  888  8888b.          *
10 *             888 d88""88b 88K      888 "88b 888  888     "88b         *
  *              888 888  888 "Y8888b. 888  888 888  888 .d888888         *
12 *             88P Y88..88P      X88 888  888 Y88b 888 888  888         *
  *              888  "Y88P"   88888P' 888  888  "Y88888 "Y888888         *
```

```
14 *             .d88P                                                          *
   *           .d88P"                              2006 Kai Uhlemann           *
16 *          888P"                                                           *
   *                                                                          *
18 * Created at: Mon Nov  7 10:58:14 EST 2005                                  *
   * System: Linux 2.6.8-2-686-smp on i686                                     *
20 *                                                                          *
   * Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.    *
22 *                                                                          *
   ****************************************************************************/
24 /*!
     \file utils.h
26   \brief utility functions used by client and/or server

28   This utility module contains a lot of functions which are used
     by both client and server. Since most of the functionality will
30   be implemented into the client due to performance issues most of
     the function will actually not be used by the server.
32  */
   /*****************************************************************************
34 *                                                                            *
   * Headers                                                                    *
36 *                                                                            *
   *****************************************************************************/
38 #include <stdio.h>
   #include <stdlib.h>
40 #include <stdarg.h>
   #include <errno.h>
42 #include <string.h>
   #include <unistd.h>
44 #include <ctype.h>
   #include <signal.h>
46 #include <arpa/inet.h>
   #include <netdb.h>
48 #include <netinet/in.h>
   #include <sys/socket.h>
50 #include <wait.h>
   #include <sys/stat.h>
52 #include <sys/types.h>
   #include <sys/ioctl.h>
54 #include <fcntl.h>
   #include <dirent.h>
56 #include <setjmp.h>
   #include <time.h>
58 #include <getopt.h>
   #include <confuse.h>
60 #include <transis/events.h>
   #include <transis/zzz_layer.h>
62 #include <transis/sim_layer.h>
   //#include <pbs_ifl.h>
64 //#include <pbs_error.h>
   #include <sys/time.h>
66 #include "data.h"
   #include "list.h"
68 #include "log.h"
   #include "msg.h"
70 /*****************************************************************************
   *                                                                            *
72 * Macro functions                                                            *
   *                                                                            *
74 *****************************************************************************/
   #define STRINGIFY(x) #x
76 #define EXPAND(x) STRINGIFY(x)
```

```
#define getstdin() readfd(STDIN_FILENO)
/*****************************************************************************
*                                                                            *
* getstdin                                                                   *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *getstdin()
 \brief simple function to get user input into a dynamic string

 function to read the stdin from command line into a buffer
 getstdin returns a fully allocated string containing STDIN
 \return fully allocated string containing STDIN
*/
/* see define section for prototype */
/*****************************************************************************
*                                                                            *
* readfd                                                                     *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *readfd(int fd)
 \brief simple function to get fd content into a dynamic string

 function to get the content from a file descriptor into a buffer
 getstdin returns a fully allocated string
 \param fd filedescriptor to read data from
 \return fully allocated string containing content of fd
*/
char *readfd (int fd);
/*****************************************************************************
*                                                                            *
* writefd                                                                    *
*                                                                            *
*****************************************************************************/
/*!
 \fn char writefd(int fd)
 \brief simple function to write buffer into fd

 function to write the content of buffer into a file descriptor
 getstdin returns 0 on success or -1 on failure
 \param fd filedescriptor to write data to
 \param *buffer the buffer holding the content
 \param length the size of the buffer
 \return 0 on success or -1 on failure
*/
int writefd (int fd, char *buffer, unsigned int  length);
/*****************************************************************************
*                                                                            *
* chkstdin                                                                   *
*                                                                            *
*****************************************************************************/
/*!
 \fn int chkstdin()
 \brief simple function to whether there is something waiting on stdin

 function to check whether input is waiting on stdin to be fetched
 getstdin returns 0 on success or -1 on failure
 \return 0 on success or -1 on failure
*/
int chkstdin();
/*****************************************************************************
*                                                                            *
```

```
140 * destroystring                                                          *
    *                                                                       *
142 ************************************************************************/
    /*!
144  \fn int destroystring(char **str)
     \brief function to destroy a dynamic string
146
     function destroys a dynamic string by freeing the allocated memory and
148  sets the pointer to NULL
     destroystring returns a EXIT_SUCCESS on success or EXIT_FAILURE
150  \param **str reference to the string
     \return EXIT_SUCCESS on success or EXIT_FAILURE
152 */
    int destroystring(char **str);
154 /************************************************************************
    *                                                                       *
156 * destroylist                                                           *
    *                                                                       *
158 ************************************************************************/
    /*!
160  \fn int destroylist(char ***listvec)
     \brief function to destroy a dynamic string list
162
     function destroys a dynamic list by freeing the allocated memory and
164  sets the pointer to NULL
     destroylist returns a EXIT_SUCCESS on success or EXIT_FAILURE
166  \param ***listvec reference to the string vector
     \return EXIT_SUCCESS on success or EXIT_FAILURE
168 */
    int destroylist(char ***listvec);
170 /************************************************************************
    *                                                                       *
172 * jidget                                                                *
    *                                                                       *
174 ************************************************************************/
    /*!
176  \fn int jidget(int value)
     \brief function to get the job id of the pbs_server to a specific value
178
     function to get the job id of the pbs_server for the next submitted job
180  to a specified value. The value is changed in the serverdb of pbs_server.
     To access the file, parts of the file structure have been extracted from
182  the torque 2.0p1 source code.
     packall returns 0 on success or -1 on failure
184  \param value the job id for the next job
     \return 0 on SUCCESS or -1 onm failure
186 */
    int jidget(char *file);
188 int jidset(char *file, int jid);
    /************************************************************************
190 *                                                                       *
    * chkregfile                                                            *
192 *                                                                       *
    ************************************************************************/
194 /*!
     \fn int chkregfile(const char* file)
196  \brief function checks wheather file is regular file

198  function to check the file whether it is a regular one, being used for
     joshua command line tools e.g. to prevent STDIN input mode
200  packall returns REGFILE if file was regulat or NREGFILE if not a file
     or iregular
202  \param *file the filename of the file to check
```

```
 \return REGFILE if file was regulat or NREGFILE if not a file
*/
int chkregfile(const char* file);
/*****************************************************************************
*                                                                            *
* gcwd                                                                       *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *gcwd();
 \brief function to get current path
 \return fully allocated string containing current path
*/
char *gcwd();
/*****************************************************************************
*                                                                            *
* cpystr                                                                     *
*                                                                            *
*****************************************************************************/
/*!
 \fn char *cpystr(char *orig);
 \brief make a copy of string
 \param *orig string to copy
 \return fully allocated string containing a copy of original
*/
char *cpystr(char *orig);
/*****************************************************************************
*                                                                            *
* Sleep                                                                      *
*                                                                            *
*****************************************************************************/
/*!
 \fn void Sleep(int sec_dlay, int usec_dlay);
 \brief alternative for sleepusiongh select
 \param sec_dlay wait for seconds
 \param usec_dlay wait for micro seconds
*/
void Sleep(int sec_dlay, int usec_dlay);
```

utils.c

```
/*****************************************************************************
* Project: JOSHUA                                                            *
* Description: utilitiy functions for all JOSHUA components                  *
* Author: Kai Uhlemann, <kai.uhlemann@nextq.org>                             *
*                                                                            *
*           888888                    888                                  *
*             "88b                    888                                  *
*              888                    888                                  *
*              888  .d88b.  .d8888b   88888b.  888  888  8888b.              *
*              888 d88""88b 88K       888 "88b 888  888     "88b             *
*              888 888  888 "Y8888b.  888  888 888  888 .d888888             *
*              88P Y88..88P      X88  888  888 Y88b 888 888  888             *
*              888  "Y88P"   88888P'  888  888  "Y88888 "Y888888             *
*            .d88P                                                           *
*          .d88P"                              2006 Kai Uhlemann             *
*         888P"                                                              *
*                                                                            *
* Created at: Mon Nov  7 10:58:14 EST 2005                                   *
* System: Linux 2.6.8-2-686-smp on i686                                      *
*                                                                            *
* Copyright (c) 2006 Oakridge National Laboratory  All rights reserved.      *
```

```
22 *                                                                            *
   ******************************************************************************/
24 /******************************************************************************
   *                                                                            *
26 * Headers                                                                    *
   *                                                                            *
28 ******************************************************************************/
   #include "utils.h"
30 /******************************************************************************
   *                                                                            *
32 * chkregfile                                                                 *
   *                                                                            *
34 ******************************************************************************/
   int chkregfile(const char *file)
36 {
     /*
38   * local data
     */
40   struct stat fbuf;

42   /* ckeck for regular file using stat */
     if(stat(file, &fbuf)==-1)
44   {
       /* if file does not exist, doesnt matter */
46     if(errno!=ENOENT)
       {
48       fprintf(stderr,"Error: stat %s\n",strerror(errno));
         exit(EXIT_FAILURE);
50     }
       return NREGFILE;
52   }
     if(S_ISREG(fbuf.st_mode))
54   {
       return REGFILE;
56   }
     else
58   {
       return NREGFILE;
60   }
   }
62 /******************************************************************************
   *                                                                            *
64 * readfd                                                                     *
   *                                                                            *
66 ******************************************************************************/
   char *readfd (int fd)
68 {
   /*
70 * local data
   */
72   int bytes=0;
     unsigned int index=0;
74   char buffer[MAX_MSG_SIZE];
     int length=MAX_MSG_SIZE-1;
76   char *content=NULL;

78   /* set memory */
     memset(buffer, '\0', length+1);
80
     for (index = 0; index < length;)
82   {
       /* Read some data. */
84     switch (bytes = read(fd, buffer + index, length - index))
```

```
    {
86    case -1:
      {
88      switch (errno)
        {
90        case EINTR:
          {
92          break;
          }
94        case EAGAIN:
          {
96          break;
          }
98        default:
          {
100         fprintf(stderr, "Warning: Unable to read from filedescriptor\n");
            return NULL;
102       }
        }
104     break;
      }
106   case 0:
      {
108     errno = EPIPE;
        if (0 == index)
110     {
          fprintf(stderr, "Warning: Unable to read from closed file descriptor\n");
112       return NULL;
        }
114     length=0;
        break;
116   }
    default:
118   {
      index += bytes;
120   }
  }
122 }

124 if(index!=0)
  {
126   /* start allocating */
    content = (char *)malloc(sizeof(char)*(index+1));
128 if(content==NULL)
    {
130   fprintf(stderr, "Error: malloc %s\n",strerror(errno));
    }
132 /* set memory */
    memset(content, '\0', index+1);
134

    strncpy(content, buffer, index);
136

    return content;
138

  }
140

  return NULL;
142 }
/*****************************************************************************
144 *                                                                           *
* writefd                                                                   *
146 *                                                                           *
*****************************************************************************/
```

```
148 int writefd (int fd, char *buffer, unsigned int  length) {
    /*
150 * local data
    */
152   int bytes=0;
      unsigned int index=0;
154
      /* check sanity */
156   if(buffer==NULL)
      {
158     return -1;
      }
160
      for (index = 0; index < length;) {
162     /* Write some data. */
        switch (bytes = write(fd, buffer + index, length - index)) {
164       case -1: {
            switch (errno) {
166           case EINTR:
              case EAGAIN: {
168             break;
              }
170           case EPIPE: {
                if (0 != index) {
172                 log_warn("Unable to write to closed file descriptor\n");
                }
174             return -2;
              }
176           default: {
                  log_warn("Unable to write to file descriptor\n");
178             return -1;
              }
180         }
            break;
182       }
          default: {
184         index += bytes;
          }
186     }
      }
188   return 0;
    }
190 /*****************************************************************************
    *                                                                           *
192 * chkstdin                                                                  *
    *                                                                           *
194 *****************************************************************************/

196 int chkstdin(){

198   /* data for select */
      fd_set fdset;
200   struct timeval tv;
      int ret;
202
      /* decide which output to read */
204   /* empty fdset */
      FD_ZERO(&fdset);
206   FD_SET(STDIN_FILENO, &fdset);
      /* dont wait for input */
208   tv.tv_sec = 0;
      tv.tv_usec = 100;
210
```

```
212   /* seclect file descriptor */
      ret=select(STDIN_FILENO+1,&fdset,NULL,NULL,&tv);
214   if(ret==-1)
      {
216     fprintf(stderr, "Error: select %s\n", strerror(errno));
        return -1;
218   }
      else
220   {
        if(ret>0)
222     {
          if(FD_ISSET(STDIN_FILENO,&fdset))
224       {
            return 0;
226       }
        }
228     else
        {
230       /* no warning whatsoever */
          return -1;
232     }
      }
234   return -1;

236
  }
238 /*****************************************************************************
  *                                                                           *
240 * destroystring                                                             *
  *                                                                           *
242 *****************************************************************************/
  int destroystring(char **str)
244 {

246   if(*str==NULL)
      {
248     return EXIT_FAILURE;
      }
250   free(*str);
      *str=NULL;
252   return EXIT_SUCCESS;

254 }
  /*****************************************************************************
256 *                                                                           *
  * destroylist                                                               *
258 *                                                                           *
  *****************************************************************************/
260 int destroylist(char ***listvec)
  {
262   /*
      * local data
264   */
      int entries=0;
266
      if(*listvec!=NULL)
268   {
        while((*listvec)[entries]!=NULL)
270     {
          destroystring(&(*listvec)[entries++]);
272     }
      }
```

```
274   /* free the list vector */
      free(*listvec);
276   *listvec=NULL;
      return EXIT_SUCCESS;
278
}
280 /*****************************************************************************
*                                                                            *
282 * jidget                                                                     *
*                                                                            *
284 *****************************************************************************/
int jidget(char *file)
286 {
  /*
288   * local data
  */
290   int fd=0,i=0;

292   if(file==NULL)
  {
294     return -1;
  }
296
  /* open file */
298   if((fd=open(file , O_RDONLY,0))== -1)
  {
300     fprintf(stderr,"Error: open %s\n",strerror(errno));
      exit(EXIT_FAILURE);
302   }

304   /* read file into struct */
      if((i=read(fd, (char *)&sv_qs, sizeof(struct server_qs)))!=sizeof(struct
306 server_qs))
  {
308     fprintf(stderr,"Error: read %s\n",strerror(errno));
      exit(EXIT_FAILURE);
310   }
      close(fd);
312
      return sv_qs.sv_jobidnumber;
314
}
316
/*****************************************************************************
318 *                                                                            *
* jidset                                                                     *
320 *                                                                            *
*****************************************************************************/
322 int jidset(char *file , int jid)
{
324   /*
  * local data
326   */
      int fd=0,i=0;
328
      if(file==NULL)
330   {
        return -1;
332   }

334   /* open file */
      if((fd=open(file , O_RDWR,0))== -1)
336   {
```

```
      fprintf(stderr,"Error: open %s\n",strerror(errno));
338     exit(EXIT_FAILURE);
    }
340
    /* read file into struct */
342 if((i=read(fd, (char *)&sv_qs, sizeof(struct server_qs)))!=sizeof(struct
server_qs))
344 {
      fprintf(stderr,"Error: read %s\n",strerror(errno));
346     exit(EXIT_FAILURE);
    }
348
    /* set the gid */
350 sv_qs.sv_jobidnumber=jid;

352 if(lseek(fd, 0, SEEK_SET)<0)
    {
354   fprintf(stderr,"Error: seek %s\n",strerror(errno));
      exit(EXIT_FAILURE);
356 }

358 /* write struct back to file */
    if((i=write(fd, &sv_qs, sizeof(struct server_qs)))!=sizeof(struct
360 server_qs))
    {
362   fprintf(stderr,"Error: write %s\n",strerror(errno));
      exit(EXIT_FAILURE);
364 }
    close(fd);
366
    return 0;
368
}
370
/*****************************************************************************
372 *                                                                           *
* gcwd                                                                      *
374 *                                                                           *
*****************************************************************************/
376 char *gcwd(){

378 #ifdef  PATH_MAX
    static int  pathmax = PATH_MAX;
380 #else
    static int  pathmax = 0;
382 #endif

384 #define PATH_MAX_GUESS  1024
    /* if PATH_MAX is indeterminate */
386 /* we're not guaranteed this is adequate */
    char *path;
388
    if (pathmax == 0)
390 {
    /* first time through */
392 errno = 0;
    if ( (pathmax = pathconf("/", _PC_PATH_MAX)) < 0)
394   {
        if (errno == 0)
396     {
    pathmax = PATH_MAX_GUESS; /* it's indeterminate */
398     }
    else
```

```
400         {
    fprintf(stderr, "Error: pathconf for _PC_PATH_MAX\n");
402           exit(EXIT_FAILURE);
          }
404       }
        else
406         pathmax++;      /* add one since it's relative to root */
    }
408
    path=(char *)malloc(sizeof(char)*(pathmax+1));
410 if(path== NULL)
    {
412   fprintf(stderr, "Error: malloc %s\n",strerror(errno));
      exit(EXIT_SUCCESS);
414 }
    if(getcwd(path, pathmax+1)==NULL)
416 {
      fprintf(stderr, "Error: getcwd %s\n",strerror(errno));
418   exit(EXIT_SUCCESS);
    }
420
    return path;
422 }
    /*************************************************************************
424 *                                                                        *
    * cpystr                                                                 *
426 *                                                                        *
    *************************************************************************/
428 char *cpystr(char *orig)
    {
430   char *copy=NULL;
      int length=0;
432
      /* check sanity of input string */
434   if(orig==NULL)
      {
436     return NULL;
      }
438   /* get the length of the input string */
      length=strlen(orig);
440
      /* start allocating */
442   copy = (char *)malloc(sizeof(char)*(length+1));
      if(copy==NULL)
444   {
        log_err("malloc %s\n",strerror(errno));
446   }
      /* set memory */
448   memset(copy, '\0', length+1);

450   strncpy(copy, orig, length);

452   return copy;
    }
454 /*************************************************************************
    *                                                                        *
456 * Sleep                                                                  *
    *                                                                        *
458 *************************************************************************/
    void Sleep(int sec_dlay, int usec_dlay)
460 {
        struct timeval tv;
462     if (sec_dlay > 0)
```

```
    {
464         time_t start = time(0);
            for (;;)
466         {
                tv.tv_sec = sec_dlay - (time(0) - start);
468             if (tv.tv_sec <= 0) break;
                tv.tv_usec = 0;
470             (void)select(0, 0, 0, 0, &tv);
            }
472     }
        if (usec_dlay > 0)
474     {
            tv.tv_sec = 0;
476         tv.tv_usec = usec_dlay;
            (void)select(0, 0, 0, 0, &tv);
478     }

480 }
```

A.3.5 misc

configure.ac

```
1 #=================================================================================#
  # Package GNU autoconf templatefile
3 # Copyright (c) 2005 Kai Uhlemann <kai.uhlemann@nextq.org>
  #
5 # - README    for general package information.
  # - INSTALL   for package install information.
7 # - COPYING   for package license information and copying conditions.
  # - AUTHORS   for package authors information.
9 # - ChangeLog for package changes information.
  #
11 # Process the '.am' file with autogen.sh or the '.in' file with 'configure' from
  # the distribution top-level directory to create the '.in' or the target file.
13 #
  #=================================================================================#
15
  # Initialize autoconf.
17 # Define and substitute package name, version and bug report e-mail. Require
  # autoconf 2.57 or higher, mark top-level source directory and config directory,
19 # and include copyright information in configure.
  AC_INIT([joshua], [0.1], [kai.uhlemann@nextq.org])
21 AC_PREREQ([2.57])
  AC_CONFIG_SRCDIR([ChangeLog])
23 AC_CONFIG_AUX_DIR([config])
  AC_COPYRIGHT([Copyright (c) 2005 Kai Uhlemann <kai.uhlemann@nextq.org>])
25
  # Initialize automake.
27 # Require automake 1.7.2 or higher, use recursive make process and compress
  # source distributions with bzip2.
29 AM_INIT_AUTOMAKE([1.7.2 subdir-objects dist-bzip2])
  AM_MAINTAINER_MODE
31
  # Initialize rpm specific specs.
33 AC_SUBST([SUMMARY], [""])
  AC_SUBST([DESCRIPTION], [""])
35 AC_SUBST([GROUP], [Development])
  AC_SUBST([LICENSE], [GPL])
```

```
37 AC_SUBST([REQUIRES], [])

39 # Define GNU source and check for C compiler, libtool, install, rpmbuild and
   # md5sum.
41 AC_GNU_SOURCE
   AC_PROG_CC
43 AM_PROG_CC_C_O
   AC_DISABLE_SHARED
45 AC_PROG_LIBTOOL
   AC_PROG_INSTALL
47 AC_CHECK_PROG([HAVE_RPMBUILD], [rpmbuild], [yes], [no])
   AC_CHECK_PROG([HAVE_MD5SUM], [md5sum], [yes], [no])
49 AM_CONDITIONAL([HAVE_RPMBUILD], [test "$HAVE_RPMBUILD" = "yes"])
   AM_CONDITIONAL([HAVE_MD5SUM], [test "$HAVE_MD5SUM" = "yes"])
51
   # Extend header and library search path for non-system installation.
53 if ! test $prefix = NONE; then
     CPPFLAGS="$CPPFLAGS -DSYSCONFDIR=\"$sysconfdir/joshua.conf\" -I$prefix/include"
55   LDFLAGS="$LDFLAGS -L$prefix/lib -Wl,-rpath -Wl,$prefix/lib"
   fi
57
   #add library path
59 LDFLAGS="$LDFLAGS -L`pwd`/lib"
   CPPFLAGS="$CPPFLAGS -DSYSCONFDIR=\"$sysconfdir/joshua.conf\""
61
   #lib dir
63 rm -f -R lib     ; mkdir lib
   rm -f -R include     ; mkdir include
65 rm -f -R docs

67 #symlink
   ln -s ../libjutils/.libs/libjutils.a lib
69 ln -s ../libjutils/utils.h include
   ln -s ../libjutils/list.h include
71 ln -s ../libjutils/data.h include
   ln -s ../libjutils/log.h include
73 ln -s ../libjutils/msg.h include

75 # List all subdirectories with Makefiles for recursive make process.
   # All core subdirectories: m4.
77 # All individual binary subdirectories:
   AC_SUBST([PKGDIRS], ["libjutils jcmd joshua jmutex"])
79
   # Check for library 'transis'.
81 AC_CHECK_LIB([transis], [zzz_Connect], [AC_SUBST([TRANSIS_LIBS], [-ltransis])],
              [AC_MSG_ERROR([Missing transis library.])])
83
   # Check for library 'confuse'.
85 AC_CHECK_LIB([confuse], [cfg_getstr], [AC_SUBST([CONFUSE_LIBS], [-lconfuse])],
              [AC_MSG_ERROR([Missing confuse library.])])
87
   # Check for library 'transis'.
89 AC_CHECK_LIB([transis], [zzz_Connect], [AC_SUBST([TRANSIS_LIBS], [-ltransis])],
              [AC_MSG_ERROR([Missing transis library.])])
91

93 # Produce output, i.e. process all '.in' files.
   # Process all core files: install manual, RPM spec, central and m4 Makefile.
95 # Process all individual binary files, such as Makefiles.
   AC_CONFIG_FILES([spec Makefile libjutils/Makefile jcmd/Makefile joshua/Makefile jmutex/
       Makefile])
97 AC_OUTPUT
```

List of Figures

List of Tables

Declaration of Authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other University.

Reading, 14-March-2006

Kai Uhlemann