SDA Perfomance Analysis

Ron Rechenmacher Eric Neilsen

March 3, 2003

1 Introduction

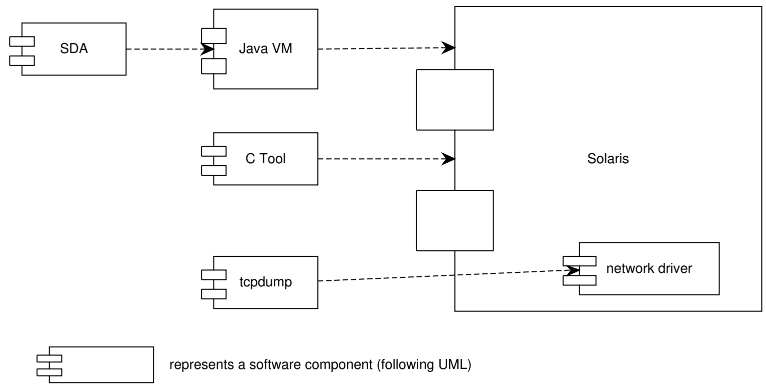
The sequenced data acquisition system, or SDA, collects data from a variety of devices that monitor accelerator performance and stores them in a database. Within a given context (called a case), the SDA reads devices at times of interest. Users specify each read time as a time after a signal (clock event or state variable transition) is received. Cases correspond to accelerator activities, and the beginning and ending times for cases are themselves tracked by SDA using signals.

Users of SDA data noticed that a few percent of SDA data were either bad or missing, and we were asked to examine this problem. This problem may have multiple causes, which could include bugs in the front end code, the use of non-atomic access to the devices causing collisions between SDA and other users, and the timing of device readings by the SDA. Because the front ends (computers dedicated to handling the instrument from which the data is collected) are real-time systems, the SDA could be implemented using a non-real-time Solaris and Java architecture. With this system design, time critical operations must be handled by the front ends. Unfortunately, some operations which may be time critical are more conveniently performed by SDA, or may not be implemented on the relevant front end. For example, if a front end cannot take a reading in response to the desired signal (or delay after that signal), the SDA must listen for the signal itself, wait any requested delay, and request a reading from the front end. An accurate understanding of the true performance characteristics of the SDA is therefore required in order to determine whether data is being collected at the correct time. Initial investigations showed this approach to be a valuable one: timing problems caused by SDA use of the JVM garbage collector were found and corrected shortly after the study began. We followed this success with a more thorough analysis of timing issues.

2 Tools

To examine SDA data collection timing issues, we developed a small set of tools. The first of these was a C program that can collect device readings in response to signals according to the same specifications that SDA does. (Its use currently requires a detailed understanding of its implementation, and is therefore not ready for use beyond its developer.) This tool can be configured using the SDA's report of its configuration, and can therefore be made to replicate the full functionality of the SDA from the point of view of the front ends. Because

Figure 1: The relationship between the software components, showing dependencies of SDA and of utilities used for testing.



it runs under the same operating system, on the same hardware, and interacts with the same front ends as the SDA, timing problems seen when using SDA but not when using the C tool can be attributed to either SDA itself or the Java platform on which it runs. Figure 1 shows this graphically.

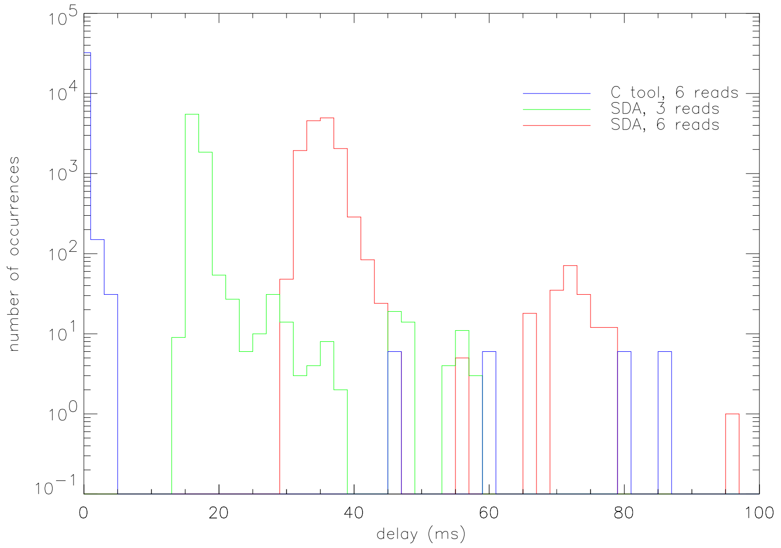
We created additional tools to provide convenient access to data recorded by SDA and other logging services from a UNIX environment. The data access tools included in SDA consist of Java utilities for simple examination of the data, export of that data to Microsoft Office formats, and an interface by which Java programs may retrieve SDA data (OSDA). Because we wished to examine the data in an environment better suited to data analysis than Microsoft Office or raw Java code, we used the Java libraries and other tools to build two utilities for extracting data from the SDA and logger databases and writing them to ASCII text files. These files can then be analyzed using a much wider variety of plotting and data analysis utilities.

3 Timing measurements

We began by using `tcpdump` on a test node to measure directly the time between a signal (in the test case, a clock event) and a request in response to that signal. We did this by running two tests on a sample SDA node. First, we compared the packet time for the signal to the system time when the signal was received. We then compared the time the signal was received by the SDA to the time at which the SDA sent a request to a front end in response.

The system time for the reception of the signal was usually within 8 ms of the packet time for that signal. However, a significant fraction of the time, the time difference could be much longer, sometimes by over 350 ms. Similarly, the typical time between reception of a signal and the sending of the resulting request was much shorter than the longest times. The SDA typically took 6 ms to respond per response it was configured to make. For example, in a test configuration where SDA was configured to respond to a clock event with

Figure 2: Time for SDA to respond to a signal with a reading request. Green represents response times for 3 signals, and red represents response times for 6. The blue represents the response time of our C tool responding to a signal with 6 requests. A Solaris network scheduling problem caused the occurrences of unusually long delays, and these delays are not seen when fix for this problem is implemented.

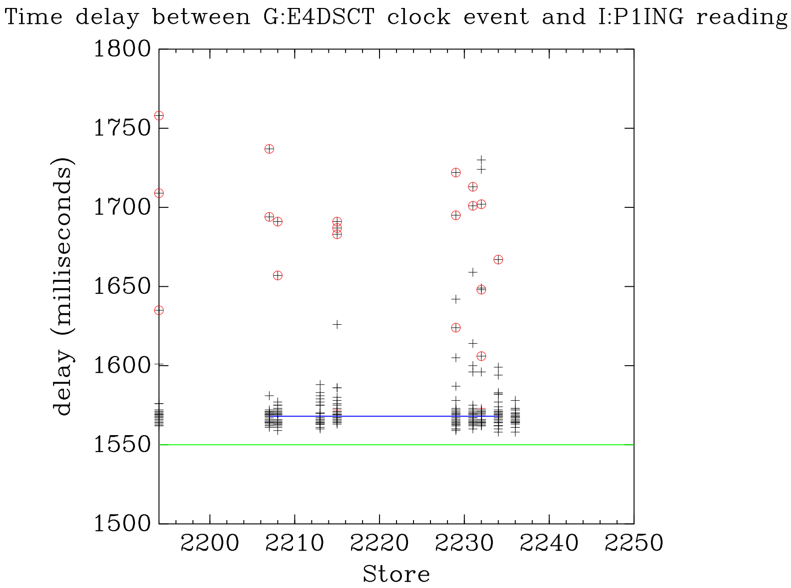


3 requests, SDA took 17 ms to respond, while it took 35 ms to respond in a configuration with 6 requests. This is a surprisingly long time given the capabilities of the hardware. The maximum time to respond was even longer, and SDA sometime took as long as 100 ms to respond. When our C testing utility was configured similarly, it usually made the 6 requests in response to the clock event in less than 2 ms, but also occasionally took much longer. Figure 2 shows histograms of the response times for the SDA when responding with 3 and 6 requests and the C tool when responding with 6 requests.

In addition to making timing measurements on a test node, we examined historical timing by comparing the collection times the SDA recorded in its database to the signal times logged for the signal on which the data collection was triggered. While this approach grants us less control and understanding of the circumstances under which the SDA displays the features we are examining, the use of actual data means that these measurements are more applicable to the typical use of the SDA. Because the logger that records the clock event times we use as reference signals is itself a Java application running on a non-real-time system, we began this test by examining the accuracy with which signal events were logged. We did this by examining the measured time differences between clock events known to occur at 5 second intervals. Again, while the measured interval was within 1 ms of 5 seconds 90% of the time, a siginicant number of intervals which differed from 5 seconds by more than 50 ms were also recorded.

We then measured the interval between the recoded data collection times for several de-

Figure 3: The time delay between the event on which the read is triggered and the time at which SDA read the I:P1ING device. The green line shows the read time requested of the SDA. The blue line shows the median of nearby data points. Red crosses show times for which SDA requested data, but an error prevented the recording of a value. Data for which SDA collected values outside the range of values (as specified in the SDA store checker configuration) expected for the device are circled in red.



vices and the recorded times of the signals which triggered data collection on those devices. The average interval was always greater than the interval requested in the SDA configuration, typically by 20 ms for the devices we measured, and individual measurements could be late by 300 ms or more. We also noted that the distribution of time delays varied over time; there is a significant improvement in the delay between October and December of 2002. We speculate that this variation was the result of changes to the configuration of the SDA.

For example, we examined the reading times I:P1ING, a device for which SDA is known to have corrupted data. Figure 3 shows the reading delays between the triggering clock event and the SDA readings. Red crosses indicate times for which SDA failed to collect any data. Points circled in red indicated times at which SDA collected data with values outside the expected range for this device, as given in the SDA store checker configuration. The plot shows that, while most collection times are 10 to 20 ms later than the specification, a significant number are much later than that, and that these collections are often suspicious.

4 Analysis and Conclusions

The order of magnitude difference in the SDA's typical response time to that of the C tool suggests there is significant inefficiency in SDA's implementation. Delays caused by these inefficiencies will be more important when many devices must be read simultaneously. We have traced the occasional dramatically longer delays which can be seen in both the C tool and in SDA to network process scheduling problem in the Solaris kernel configuration; this problem is well understood, and the long delays disappear after a fix is implemented.

Any data collection where there are strict timing requirements must be handled by the front ends in order to make timed readings reliable with the current SDA. However, some changes and optimizations may help the current system. These include:

- tweaking the SDA application code, particularly adopting a collection algorithm close to that used in our C tool (The current algorithm uses a large number of threads, which may significantly reduce the performance of the system. The recent addition of `nio` to the Java class libraries can facilitate this change.);
- determining quantitative performance requirements for the SDA based on the current and expected future needs of its users;
- creating, documenting, and following a formal process for software development, including explicit documentation of system requirements, a process for testing whether the current system meets those specifications (we note that our C tool has already been used effectively to find problems in the front ends, and could be adopted for use as part of a front end testing platform), tracking of changes in performance and functionality in each version using consistent quantitative performance metrics and regression testing, and the scrupulous use of the accelerator controls bug tracking system (currently installed and working, but unused) to document and track all known bugs and requested changes;
- further adjustment to the operating system, including modifying the Solaris kernel configuration so that the system priority of streams is raised above anything that might interrupt them (when such a reconfiguration was implemented, our C tool could reliably respond to signals with 6 requests in less than 2 ms); and
- using atomic operations for device readings to prevent collisions between SDA readings and requests by other device users. This may require supplementing the ACNET protocol and improving some front end software.

A comparison between the current performance and that which is necessary for operation is needed before a reasonable plan for future work can be made. We do not believe there are established specifications for SDA's required performance, but individuals questioned expect that the system read devices at 15Hz, or faster than 66 ms. Provided the network scheduling fix is implemented, the SDA should be able to achieve this provided it is never required to read more than 10 devices simultaneously. The comparison between the required and achieved performance must consider not only the current required response time, but also the potential number of simultaneous device reading requests and the required response times of possible additions.