Towards a Specification for Measuring Red Storm Reliability, Availability, and Serviceability (RAS)

Jon Stearley <jrstear@sandia.gov>
Sandia National Laboratories*
P.O. Box 5800
Albuquerque, New Mexico 87185-0817

17th May 2005

Abstract

The absence of agreed definitions and metrics for supercomputer RAS obscures meaningful discussion of the issues involved, hinders their solution, and increases total system cost. Seeking to foster a common basis for communication about supercomputer RAS, [1] proposed a general system state model, definitions, and measurements based on the SEMI-E10 specification [2] used in the semiconductor manufacturing industry. This document enumerates the platform-specific details necessary to apply that general framework to the Red Storm system at Sandia National Laboratories. Familiarity with [1] is a strong prerequisite for understanding of this document, as is familiarity with the Red Storm RAS subsystem [1] (although to a much lesser degree). Given the current pre-production status of Red Storm, this document does not specify actual policy or practice, but rather proposes a framework by which to measure RAS performance on Red Storm.

Keywords

Reliability, Availability, Serviceability, RAS, Metrics

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

[1] Detailed description of the Red Storm RAS subsystem is beyond the scope of this document; interested readers should consult the extensive Cray documentation on this subject.

1 Preliminary Details

Our RAS definitions [1] depend heavily upon the "intended function" of the item being considered. It is therefore appropriate to begin by stating the functions of the Red Storm system. Overall, Red Storm's function is to quickly perform large-scale computations for *production* users - therefore this document largely focuses on *production* RAS performance (e.g. from the perspective of *production* users). The system (or "section"[2]) is said to be "up" when this function is manifested via the complete set of services described in Table 1.

2 State Model

This section enumerates the specific Red Storm criteria for determining each of the six basic states depicted in Figure 1 (provided here for convenient reference, see [1] for a detailed discussion). All equipment (at both a system and component granularity) at all time falls into one of the six basic states. Tracking of state at a section granularity is done via manual records[3], whereas component states can be accomplished by periodic queries (see Table 2) or post-processing of historical logs (e.g. cpa.log and/or event.log).

[2] Please note that "section" and "system" are used interchangeably in this document except where specifically stated otherwise.

[3] It is expected that the AIRS [3] database will house these manual section state records, e.g. the time at which the system entered a scheduled downtime, or was made available to users, etc.

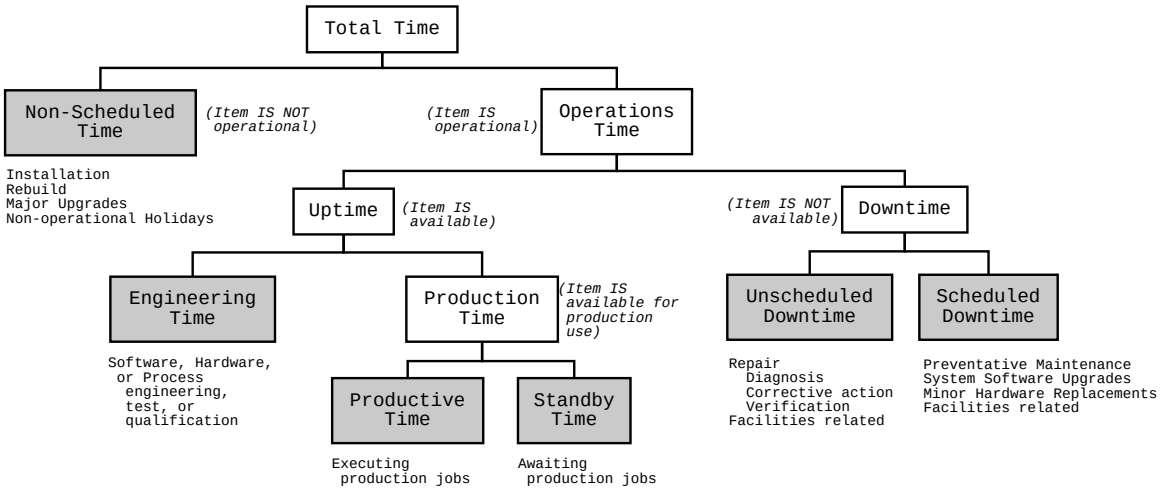

Figure 1: Hierarchy of Equipment States (basic states in gray)

No distinction is made between batch and non-batch ("interactive") nodes in tracking their state.

2.1 Productive State

A *node* is counted as being in a productive state when it is available[4] for use by only one user. The low-level criteria for this state is that the node's processor_id is present in the SystemDataBase (SDB) partition_allocation table. This occurs upon a PBS job being executed (prior to the execution of yod), or upon yod execution (for non-batch "interactive" jobs). Note that this criteria results in the shell-execution and cleanup job stages being included in production time.

Note that when a *section* is dedicated for use by a single user ("dedicated time"), all components in that section will be counted as being in a productive state - regardless of whether the user is actually running jobs or not - this is the only case for which productive state will be tracked at a section granularity.

[4] "Available" means that an item is in a condition to perform its intended function upon demand [1].

2.2 Standby State

A *node* is counted as being in a standby state when it is available for use by more than one user, but is not in a productive state due to workload conditions (e.g. there are no jobs are in the queue, or the node is being reserved for a large job which has not started yet, etc. The low-level criteria for this state is that its SDB processor_status is "up" and its processor_id is NOT present in the partition_allocation table.
Tracking of standby state at a *section* level is not expected to be necessary (although if it happens this will be visible via the utilization metrics).

2.3 Engineering State

The time when an item (e.g. node or system) is available to system engineering users (e.g. for system software and process engineering and testing, application porting and initial performance characterizations, etc). Using ASC Availability Status terminology, to date Red Storm has been in alpha and beta availability statuses, which in the model proposed herein has been considered as engineering time. This state is tracked only at a *section* granularity (not per-node, thus there is no low-level criteria given for this state).

Service Name	Description
login	Users can log in to the system.
compile	Users can compile applications.
job	Batch and interactive jobs work (submission, wait, shell-execution, application-execution, and cleanup [1]) work correctly, as are all batch queue functions (jobs can be submitted, queried, removed, and are being appropriately scheduled and executed).
io	Users and jobs can utilize the high performance file system.
scale	At least a certain number of nodes are up (e.g. 95% of the nodes in the section).

Table 1: Critical System Services

2.4 Scheduled Downtime State

The time (scheduled downtime) when an item is not available due to planned events (e.g preventative maintenance, hardware or software upgrades, power or cooling related, etc). Tracking of this state is at a *section* granularity only.

2.5 Unscheduled Downtime State

The time (unscheduled downtime) when an item is not available due to unplanned events (e.g. needs repair, is power or cooling related, etc). The low-level criteria for determining that a *node* is in this state is that its status in the SDB processor table is "down", or is configured to be present in the section but does not appear at all in the SDB. For example, if the red section is configured to contain 7500 nodes but only 7450 have status "up" in the SDB, the remaining 50 nodes are said to be in an unscheduled downtime state.

2.6 Non-scheduled State

The time (non-scheduled time) when an item is not expected to be operational (e.g. initial installation, significant rebuilds, etc). Tracking of this system state will be done manually (if at all - it is of course hoped that this state is not necessary during the useful life of the Red Storm system).

3 Failures and Interrupts

This section describes how we will categorize undesired status transitions. The general rule is that failures regard items (e.g. nodes) whereas interrupts regard work (e.g. jobs).

3.1 System Failure

A system failure is an event requiring that *the system* enter an unscheduled downtime state before *any component* may transition into a productive status. For example, the whole system must be rebooted before new jobs can execute. As this is a system event its occurrence must be recorded manually, with the time of the event being the time at which the above condition became true (rather than the time that the repairs began taking place).

3.2 Service Interrupt

A section will be considered "up" (completely working) only when a full set of services is working (see Table 1). Any time the section is in a production time status but any of the critical services is not working is called a service interrupt. This is expected to be tracked manually, however it may be possible to establish low-level criteria to identify this event if desired (e.g. less than 95% of nodes are up, pbs_scheduler process dies, high performance file system disks fail, etc). If a useful subset of services remain functional after a service interrupt, the system is said to be operating in a "degraded" mode.

A system failure results in a service interrupt, but a service interrupt may not always imply a system failure (e.g. only a subset of services are unavailable, and can be restored without interrupting the working subset of services).

3.3 Job Interrupt

Red Storm provides an excellent facility by which to identify those jobs which are interrupted due to component failure, via the SDB accounting database includes yod_err and yod_info fields which are set when jobs terminate abnormally. Furthermore, when a job is terminated by the RAS subsystem (e.g. in the event of a component failure), this is noted in the yod_info field. The intent is to identify job interrupts which are cause by component failures. However, if SIGKILL is used (e.g. by an ordinary user) to terminate the job, the RAS subsystem must clean up the remains of the job, and thus the event would be counted as a (component-failure induced) job interrupt rather than appropriate response to user behavior. The low-level criteria for this event is that the cleaned_by field of the SDB job_accounting table is set to "ras".

3.4 Node Failures

A node failure is the transition of any node into an unexpected downtime status. Additional details are necessary according to the type of node.

3.4.1 Compute Node

Any time a compute node unexpectedly becomes unable to execute production jobs is counted as a node failure. The low-level criteria for this is the setting of the node's status in the SDB processor table to "down". The only possible preceding node status settings are "up" (e.g. the node is in standby or productive state) or "suspect" (e.g. the node was in a productive state, and was then flagged as "suspect" by yod during job startup) - thus any transition to "down" constitutes a compute node failure. The SDB status is used instead of CentralDataRepository (CDR) status in order to most closely match the production user's perspective - and node failures are exposed to users via SDB clients (yod and PBS). Secondly, the SDB provides SQL query of state, whereas CDR access is via command line or C-library only (as implemented at time of this writing, there is some discussion that this may change).

3.4.2 Login Node

A login node's functions are more varied than a compute node's. However, we will use the same low-level criteria as described for compute node above. As such, we do not distinguish between which login node function failed (login, compile, job submission, parallel file system access, etc).

3.4.3 I/O Node

An I/O node's function is to provide access to the parallel file system. I/O node status is not currently captured in the SDB, so the low-level criteria used to determine an I/O node failure will be the unexpected transition of the node's status in the CDR to "down". I am unable to provide exact low-level criteria for this event at this time.

3.5 Seastar Failure

Seastar (Red Storm high-speed interconnect devices) are monitored via a "portals heartbeat" memory address being incremented by the seastar and monitored via L0[5] nodes) - any time this heartbeat *stops* would be counted as a seastar failure. I am unable to provide low-level criteria for this event at this time.

4 Measurements

This section lists metrics believed to be meaningful towards understanding Red Storm RAS performance, ordered by decreasing importance. Consult [1] for detailed equations and discussion, but it is worth emphasizing here that whereas (T)ime based metrics are common, (N)odehour metrics provide additional insight because they include workload information (failures are more a function of workload than time).

Whereas [1] uses only general terms in equations, in this document I will define some variables towards a possible implementation of the below metrics. Given a time period $\mathbf{T}$ where $t_i$indicates the time of measurement and i ranges from 1 to l, let $\mathbf{U}$ be a time series where u_i indicates the number of "up" nodes at time t_i, and $\mathbf{P}$ and $\mathbf{S}$ be series indicating the number of productive-state and standby-state nodes respectively. Furthermore, let $\mathbf{D}$ indicate the number of nodes which *transitioned* from an

[5] L0 nodes are embedded Linux nodes which are part of the Red Storm RAS subsystem. Again, readers are referred to the extensive Cray documentation of this subsystem for more details.

Item	State	Variable	Value
nodes	up	u_i	= SELECT COUNT(processor_id) FROM processor WHERE status='up';
	productive	p_i	= SELECT COUNT(DISTINCT processor_id) FROM partition_allocation;
	standby	s_i	$= u_i - p_i$
	downs	d_i	$= (u_i - u_{i+1}) > 0 \,?\, (u_i - u_{i+1}) \,:\, 0$ (an estimate only - see text!)
jobs	interrupted	j_i	= SELECT COUNT(partition_id) FROM job_accounting WHERE destroy_time>=t_{i-1}AND cleaned_by='ras';

Table 2: Counting compute and login nodes at time t_i

up to a down status during time period t_i. Table 2 describes the values of these series (all series are of length l). A first-order estimate of $\mathbf{D}$ can be obtained by tracking changes in $\mathbf{U}$, however nodes which also *transition back* to an up status during the time period will be missed by this estimate. A more precise assignment of $\mathbf{D}$ could be obtained by parsing the RAS subsystem event.log. Additionally, assignment of $\mathbf{P}$ as described will include interrupted jobs as counting towards productive time (and all subsequent metrics) - if this is not desired, $\mathbf{P}$ could be modified by subtracting the job size from all affected p_i (all time periods during the job which was interrupted). Values could also be obtained by processing of event logs, but this document uses SQL queries because they are simpler to implement, and not expected to unacceptably burden the system.

The above variables and below metrics do not constitute challenging mathematics, but rather a well-defined means of quantifying RAS performance. The variables as described above do however lend themselves to simple expressions below, and allow for the study of events over time. For example one could plot node and job failures versus time and examine their distribution - do they exhibit a constant failure rate (thus justifying an exponential random variable model for the calculation of reliability), or not (perhaps, a Poisson random variable is more appropriate), etc [1].

Not all the variables below are precisely described, and many of them refer to the aforementioned manual records described in Section 2. This document is a work in progress and will be updated as actual tracking of state and calculation of metrics is performed. The first tracking of Red Storm state using this model is expected to occur in June, when early production users begin using the red section.

4.1 Reliability

4.1.1 Mean Time and Nodehours Between System Failures

Mean time between system failures (see section 3.1) is a primary system reliability metric, calculated as

$$MTBF_{System} = \frac{production\,time}{number\,of\,system\,failures}.$$

This is related to requirement 5.14 of [4] which states that "the need to reboot the system shall be greater than 100 hours of continuous operation ... 99% of resources available."

Mean Nodehours Between System Failures is similar to $MTBF_{System}$, but provides additional information as it is a function of workload rather than raw time. It is calculated as

$$MNBF_{System} = \frac{productive\,nodehours}{number\,of\,system\,failures}.$$

4.1.2 Mean Time and Nodehours Between Node Failures

Mean time between node failures (see section 3.1) is a primary node reliability metric, calculated as

$$MTBF_{Node} = \frac{production\,time}{number\,of\,node\,failures} = \frac{t_l - t_1}{sum(\mathbf{F})}.$$

Mean Nodehours Between System Failures is similar to $MTBF_{Node}$, but provides additional information as it is a function of workload rather than raw time. It is calculated as

$$MNBF_{Node} = \frac{productive\,nodehours}{number\,of\,node\,failures} = \frac{sum(\mathbf{P})}{sum(\mathbf{F})}.$$

4.1.3 Mean Time and Nodehours Between Job Interrupts

This is related to requirement 5.13 of [4] which states that job interrupts "MTBI for the full system shall be greater than 50 hours for continuous operation of the full system on a single application".

$$MTBI_{Job} = \frac{production\,time}{number\,of\,job\,interrupts} = \frac{t_l - t_1}{sum(\mathbf{J})}$$

$$MNBI_{Job} = \frac{productive\,nodehours}{number\,of\,job\,interrupts} = \frac{sum(\mathbf{P})}{sum(\mathbf{J})}$$

It is worth noting that the total number of jobs is not present in these calculations. Similar metrics which include the total number of jobs may also be useful towards understanding the impact of the runtime software on the reliability of the system (e.g. the higher the total number of job executed, the more opportunity for runtime job start/stop software bugs to manifest themselves in job interrupts).

4.1.4 Mean Time and Nodehours Between Service Interrupts

Service interrupts are of principal concern to users - these metrics convey the average time and productive work between such events. They are aggregate metrics, affected by both scheduled and unscheduled service interrupts.

$$MTBI_{Service} = \frac{production\,time}{number\,of\,service\,interrupts}$$

$$MNBI_{Service} = \frac{productive\,nodehours}{number\,of\,service\,interrupts}$$

4.2 Availability

4.2.1 Total Availability

Total availability is an important availability metric, calculated as

$$Total\,Availability_{System}(\%) = \frac{uptime}{total\,time} * 100.$$

If engineering and non-scheduled time is not negligible (as expected), production availability is a useful supplement, calculated as

$$Production\,Availability_{System}(\%) = \frac{production\,time}{operations\,time} * 100.$$

4.3 Serviceability

4.3.1 Mean Time To Boot System

The importance of tracking mean time to boot the system on an ongoing basis will largely depend on how often system boots are required, and is calculated as

$$MTTB = \frac{total\,time\,spent\,booting}{number\,of\,boot\,cycles}.$$

This is related to requirement 5.11 of [4] which specifies that "a full system reboot of the classified or unclassified sections from a clean shutdown and without a disk system fsck shall take less than 15 minutes."

4.3.2 Mean Nodehours To Repair

This calculation measures the average computational ability lost per failure. Due to the use of nodehours, this metric is sensitive to partial as well as complete system failures.

$$MNTR = \frac{unscheduled\,downtime\,nodehours}{number\,of\,failures}$$

4.4 Other Miscellaneous Measurements

The following are not RAS metrics, but are included here for completeness.

4.4.1 Total System Utilization

Total utilization is very common aggregate metric of RAS, job workload, and queuing policy information, calculated as

$$Total\,Utilization_{System}(\%) = \frac{productive\,time}{total\,time} * 100 = \frac{sum(\mathbf{P})}{t_l - t_1}.$$

4.4.2 Capability Usage Performance

A DOE requirement for Red Storm is that 80% of the computational time be performed by jobs using at least 40% of the total possible nodes. We intend to calculate this per section (red, black) and size (25% or 75% of total nodes) , as

$$Capability Usage Performance_{40\%} = \frac{productive\ nodehours\ by\ jobs\ at\ least\ 40\%\ of\ section}{total\ productive\ nodehours}.$$

It is unclear at this time how to aggregate these per section and size metrics, as well as inclusion of "jumbo mode" runs.

4.4.3 Scheduled System Availability (%)

This calculation measures how fully uptime expectations are met during a time period. The key feature of this metric is that quantitative expectations exist (e.g. uptime and downtime schedules are set at the beginning of the time period).

$$Scheduled\ Availability_{System}(\%) = \frac{uptime - downtime}{scheduled\ uptime} * 100$$

5 Conclusions

In order to have effective communication about Red Storm RAS performance, we must agree on what the terms we use mean. In order to have quantitative understanding of Red Storm RAS performance, we must enumerate precise states (and the conditions necessary and sufficient to determine them) and equations for quantitative metrics. The union of [1] and this document forms this set. It is hoped that these documents prove useful in facilitating the ongoing evaluation and improvement of Red Storm RAS performance, and in assessing the value of the extensive Red Storm RAS subsystem.

Acknowledgments

Sue Kelly, Bob Ballance, and Jim Ang provided significant input and/or review of this document - thank you very much!

Revision

Revision 1.10 of this document appears in the Proceedings of the 2005 Cray Users Group Meeting. This document will be revised according to the actual tracking of Red Storm RAS performance. This is $Revision: 1.10 $, $Date: 2005/05/18 05:17:56 $.

References

[1] Jon Stearley. Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS). In *Proceedings of the 2005 Linux Clusters Institute Conference*, 2005. http://www.cs.sandia.gov/~jrstear/ras.

[2] Semiconductor Equipment and Materials International. Specification for definition and measurement of equipment reliability, availability, and maintainability. SEMI E10-0304, 1986, 2004.

[3] Robert Ballance, Jared Galbraith, and Roy Heimbach. Supercomputing center management using AIRS. In *Proceedings of the 2003 Linux Clusters Institute Conference*, 2003.

[4] ASC Red Storm acceptance test plan. Cray and Sandia National Laboratories internal document.