RAS Framework Engine Prototype

A Dissertation

Submitted In Partial Fulfilment Of

The Requirements For

Masters of Science

In

NETWORK CENTERED COMPUTING,

E-Business

in the

FACULTY OF SCIENCE

THE UNIVERSITY OF READING

by

Antonina Litvinova

22/09/2009

Oak Ridge National Laboratory Supervisor:
Dr. Christian Engelmann

University of Tennessee Supervisor:
Dr. George Bosilca

Director of the course at the University of Reading:
Professor Dr. Vassil Alexandrov

ACKNOWLEDGEMENTS

"There is a light at the end of the tunnel".

I would like to thank the Director of my course Professor Dr. Vassil Alexandrov for giving me the opportunity to do my Masters research at Oak Ridge National Laboratory (ORNL).

I enjoyed studying at the University of Reading in the UK with my professors: Dr. Garry Smith, Dr. Hugo Mills, Dr. Ronan Jamieson, Research Assistant Nia Alexandrov, PhD. student Andrea Weise, Professor Dr. Mark Baker, Dr. Jordi Garcia Almiñana, PhD. student Asaad Moosa; and doing my project at ORNL, USA.

Writing a thesis was a big challenge for me to make a first attempt at serious research. The thesis describes my understanding of the whole project with my work and opinions.

I would like to thank: my supervisor and mentor Dr. Christian Engelmann for his professional day-to-day supervision, and his team (at ORNL): Dr. Stephan L. Scott, Dr. Geoffroy Vallee, PhD. student Thomas Naughton, Dr. Hong Ong, Anand Tikotekar, Cindy Sonewald, Dr. George Ostrouchov; Dr. Houssain Kettani and students from Puerto Rico for related work.

I would like to thank my supervisor at the University of Tennessee (UT), Dr. George Bosilca and his team: Dr. Aurelien Bouteiller, Dr. Thomas Herault and Dr. Pierre Lemarinier for great support and critical review, and Wendy Syer for the welcoming programme.

I would like to thank my family for their continuous support and all my friends from the USA, the UK, Lithuania, Italy, Greece, Netherlands, Estonia, Spain, France, Russia and Ukraine (I apologise for not translating to every language because of the lack of space).

This RAS Framework Engine Prototype research was sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

Very many people were involved in helping me (to different extents) to make this project possible, enjoyable, and successful in order to complete the Masters programme. I am grateful to YOU all.

For more information please contact:

Antonina Litvinova a.litvinova@student.reading.ac.uk,

Christian Engelmann engelmannc@ornl.gov

George Bosilca bosilca@eecs.utk.edu

ABSTRACT

Extreme high performance computing (HPC) systems constantly increase in scale from a few thousands of processors cores to thousands of thousands of processors cores and beyond. However their system mean-time to interrupt decreases according. The current approach of fault tolerance in HPC is checkpoint/restart, i.e. a method based on recovery from experienced failures. However checkpoint/restart cannot deal with errors in the same efficient way anymore, because of HPC systems modification. For example, increasing error rates, increasing aggregate memory, and not proportionally increasing input/output capabilities. The recently introduced concept is proactive fault tolerance which avoids experiencing failures through preventative measures. Proactive fault tolerance uses migration which is an emerging technology that prevents failures on HPC systems by migrating applications or application parts away from a node that is deteriorating to a spare node. This thesis discusses work conducted at ORNL to develop a Proactive Fault Tolerance Framework Engine Prototype for HPC systems with high reliability, availability and serviceability. The prototype performs environmental system monitoring, system event logging, parallel job monitoring and system resource monitoring in order to analyse HPC system reliability and to perform fault avoidance through a migration.

Contents

List of Figures

List of Tables

1. INTRODUCTION

How to design an extremely powerful high-performance computing (HPC) system with fast computations and at the same time with never having interruptions? This question is teasing computer science researchers since the first computer was developed. Progress is not idle when it relates to developments and engineering in computer science. There are improvements every day to discover something new or something better or extraordinary.

In 2008 scientists reached the petaflops era in a comparably short period of time, because architecture, network and processor design constantly change. Only 5 years back HPC systems were able to compute in 3.2 teraflops [1].

The ranking of the Top 6 list of the most powerful Supercomputers in the world in June 2009 [2]:

1. Roadrunner at Los Alamos National Laboratory (LANL), USA. It has a difficult and specially designed architecture.
2. Jaguar: Cray XT5 at Oak Ridge National Laboratory (ORNL), USA.
3. Junene: Blue Gene/P at Forschungszentrum Juelich (FZJ), Germany.
4. Pleiades: SGI at Ames Research Centre
5. Blue Gene/L at Lawrence Livermore National laboratory (LLNL), USA.
6. Kraken: Cray XT5 belongs to the University of Tennessee, USA (located at ORNL). Kraken is the most powerful academic supercomputer in the world.

A HPC system is a powerful and important tool, able to open new horizons for computer scientists in making computations, understanding problems, making visualizations of climate dynamics and multimedia cartoons, making models in molecular chemistry, biology, fusion, energy, looking inside the human body and much more (astrophysics, physics, computer science, engineering) without making physical objects in the real world [4]. These innovations make the wheel of science spin with powerful and productive results.

In 2009, Oak Ridge National Laboratory allocated the following projects on its HPC systems [3]:

- Astrophysics;
- Biology;
- Chemistry;
- Climate;
- Computer science;
- Engineering;
- Fusion;
- Physics.

The results of these projects will give better understanding for scientists in theory and for humans in practical implementations. The allocated time for each of these projects can reach 60 million hours, but it may take much longer if a failure occurs.

HPC systems constantly increase in scale from a few thousands of processor cores to thousands of thousands of processor cores and beyond. The systems performance capabilities increase as well. But the performance does not increase proportionality to systems powerfulness. This can be seen in a decline in the proportion of mean-time to interrupt (MTTI). While the systems continue to increase in scale their architecture changes. The system parts are taken away to form new components such as head node, compute nodes and input/output nodes, separate storage. Other parts are taken away completely because they reduce reliability of the system or there is no need for them, for example, sound and graphics cards.

As the overall number of components increases, failures will occur at a much higher rate than on older systems. For example, if the annual failure rate is less than 1% on a system with a hundred processors the rate will be higher on a system with a thousand of processors. Failures are always causing problems. It is easy to see that the systems need support in dealing with failures. Having a failure in a system means having lower performance.

Failures are different from each other. Failures can be software failures, hardware failures, human errors, network failures and unknown failures. A failure is a result of a covered or uncovered fault. If there is a fault or many faults on a system, it does not necessarily mean that the system will fail now but that is might fail eventually. In the event of a failure, Fault Tolerance (FT) mechanism is designed for HPC systems. Fault tolerance mechanism helps keep computations running and finishing. There are several FT mechanisms but the most used state of practice is Reactive FT. This approach of FT in a HPC system is based on checkpoint/restart, i.e. a method on recovery from experienced failures.

Checkpoint/restart is a process of stopping the applications and making a snapshot of the current state, saving the snapshot and resuming the computations. The applications are checkpointed at predefined intervals. Whenever a failure occurs the system rolls back to the previous checkpoint or sometimes to the very first checkpoint to restart a part of an application or the whole application in order to continue computing and finish the process. However Reactive FT is not able to handle failures in the same efficient way as before because of the increasing number of failures. If this approach will continue to be used in the near future the checkpointing process will take the time of the computing, i.e. the same time will be spent on checkpointing, as doing the real computations.

The recently introduced approach [22] is Proactive FT. The new FT aim is not to wait until a failure occurs but to avoid it by using pre-emptive methods such as predicting a failure and making a migration. In this case the failure will be avoided and not experienced. Proactive FT uses migration which is an emerging technology that prevents failures on HPC systems by migrating application or application parts away from a node that is deteriorating to a spare node. But this approach has some limitations, for example the migration must be completed before a failure actually occurs and some failures, as double bit flips unpredictable. New thoughts [24] were put to improve this approach to combine Proactive FT with Reactive to have a so called Holistic FT. The last approach is based on a RAS Framework.

1.1 Thesis Statement

The proposed solution is to investigate Proactive FT for HPC systems and find available HPC fault tolerance solutions, by developing a proactive fault tolerance framework. RAS Framework is a proposed solution i.e. a mechanism of handling failures in HPC systems to improve reliability, availability and serviceability.

This framework is based on:

1. Proactive FT using pre-emptive migration;
2. Reactive FT using checkpoint/restart;
3. Reliability Analysis for making predictions;
4. Other necessary parts can be also added.

1.2 Contribution

Contribution was made towards a design of a RAS Framework Engine Prototype. The prototype is capable of (1) accumulating environmental monitoring data, system log data and parallel job data, (2) analysing the accumulated data, and (3) making informed decisions on migrating application parts away from compute nodes that are about to fail.

1. Aim: create interfaces between software
   Objectives: looking at the software content
   Hypothesis: aggregated data will be stored

2. Aim: create processes for continuous data storage
   Objectives: looking at environment (Operating System)
   Hypothesis: aggregated data will be stored continuously

3. Aim: have collected data
   Objectives: create database
   Hypothesis: to use collected data for failure prediction

1.3 Summary

Since the HPC systems increase in scale failures are increasing. The current state of practice of dealing with failure using Reactive FT, is not capable of dealing with failures in the same efficient way as before. This project aims to analyse the current state of the art by designing and developing a RAS Framework Engine Prototype. This chapter clearly states aims, objectives and hypothesis of the research.

1.4 Thesis Structure

While this chapter introduces the project which is based on developing a RAS Framework Engine Prototype for dealing with failure in HPC systems, gives aims, objectives and hypothesis, the rest of the thesis is structured as follows: Chapter 2 describes the background and previous work, Chapter 3 tells about Preliminary Framework Design, Chapter 4 gives Implementation Strategies and Chapter 5 shows Detailed RAS Framework Engine Prototype design. The chapter 6 describes conclusion with future work and critical review. The chapters 7 consists of list of References and chapter 8 includes Appendixes with CD, users manual and other relevant documentations that can be found at the end of the thesis.

2. REVIEW OF LITERATURE

“If mathematics is the language of science, computation is its workhorse.” [1]

This chapter introduces the past research, studies and developments of Proactive Fault Tolerance found in journals, books and published papers. It also provides an overview of computer science and explains related work.

2.1 Background

2.1.1 Supercomputers

An HPC system is a supercomputer designed to do high level computations. Compared to ordinary desktop machines, large-scale HPC systems do not have compute-node, local disks as they reduce reliability and they do not have graphics and sound cards as they are not needed. The systems have a complicated design because they are designed to make faster computations. A typical HPC system consists of a head node (i.e. computer), compute nodes, input/output (I/O) nodes and a file system (FS), connected through a high speed network.

In 2009 the most of 6 powerful Supercomputers in the world are [2]:

Rank	Site	Computer	Country	Cores	R max (Tflops)	R peak (Fflops)	Power (KW)
1	LANL	Roadrunner	USA	129600	1105	1456.7	2483.47
2	ORNL	Jaguar: Cray XT5	USA	150152	1059	1381.4	6950.6
3	FZJ	Jugene: BlueGene/P	Germany	294912	825.5	1002.7	2268
4	Ames Research Centre	Pleiades: SGI	USA	51200	487.01	608.83	2090
5	LLNL	BlueGene/L	USA	212992	478.2	596.38	2329.6
6	UT	Kraken: Cray XT5	USA	66000	463.3	607.2	No data provided

Table 2.1 Top 6 Most Powerful Supercomputers

The table 2.1 consist of Supercomputers names and their locations, number of node cores (each node has multiple core architecture). R max represents maximal LINPACK performance achieved. R peak – performance of theoretical peak. And the last column shows the power used to run the machine per year in kilowatts.

There are powerful computing resources at ORNL [3]: Jaguar, Kraken, Smoky is a 80 node Linux cluster, Lens is a 32 node Linux cluster dedicated to data analysis and high-end visualization on a high resolution screen called EVEREST, and Eugene.

One of these powerful systems is Jaguar which is located at ORNL. In 2008 Jaguar consisted of two systems: Cray XT5 with *petaflops* and Cray XT4 with *teraflops*. XT5 and XT4 can make 1.4 quadrillion mathematical calculations per second and 263 trillion calculations per second respectively. The overall system has 181 000 processing cores connected internally with Cray's *Seastar2+* network and combined into a single network using an *InfiniBand* that links each piece to the *Spider* FS [3].

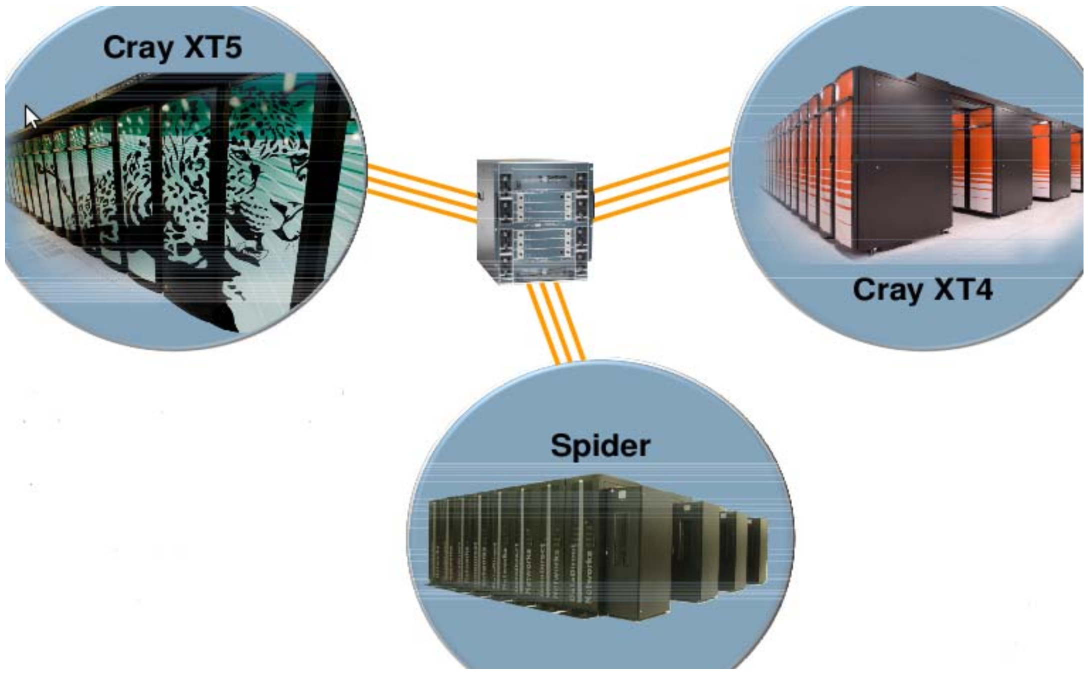


Figure 2.1 Jaguar System Components [5]

Jaguar specifications [3]:

- The system consists of cabinets and each cabinet consists of boards. Each board has 4 nodes.
- Jaguar runs SuSE Linux Operating System (OS). The OS joins the system services, networking software, communications, I/O, mathematical libraries, compilers, debuggers and performance tools to form the Cray Linux Environment. Jaguar supports Message Passing Interface (MPI), OpenMP, Shared Memory (SHMEM) and Partitioned Global Address Space (PGAS) programming models.
- Jaguar has a cooling system to control the heat of the system.
- Spider FS is based on *Lustre* FS.
- Scalable I/O Network. Networking capability is in parallel to insure accurate, high-speed data transfer.
- High Performance Storage System is for archival data storage.

Another example of an HPC system is Roadrunner which is located at LANL. The system was specifically designed to provide high computational performance within tolerable cost and power budgets. Roadrunner is a heterogeneous system containing microprocessors and has accelerators with special purpose. Each node consists of 3 blades. All triblades are interconnected using InfiniBand to model the Roadrunner system [6].

Not all scientific sites can afford to have a high powerful supercomputer or there is no need for one, i.e. doing software tests. In this case clusters and grids are used. A *cluster* is a group of two or more computer linked together in order to have greater processing power. For example in 2003 Virginia Tech University, USA was able to build a powerful Apple G5 cluster which consisted of 1100 G5 computers with 2200 processors. This cluster was ranked as the 3rd most powerful machine in the world [73].

2.1.2 High Performance Computing

Historically scientific HPC is based on *parallel* and *distributed computing*, i.e. the computational problem is solved using more than one processor [7]. Each of these types of computer is in itself, is an extreme of concurrent computing, while there are combinations in-between. In *distributed computing*, a problem is divided into many tasks, each of which is solved by one computer with its own private memory and the information is exchanged by passing messages between processors. In *parallel computing*, on the other hand, while the multiple processing elements have access to a shared memory which is used to exchange information between processors, the problem is broken into independent parts and each processing element executes its part of the algorithm simultaneously with the others [8].

It is often difficult to make a clear distinction between parallel and distributed computing. The processors in a typical distributed system run concurrently in parallel. Parallel computing may be visualized as a tightly-coupled form of distributed computing, and distributed computing may be visualized as a loosely-coupled form of parallel computing [9, 10].

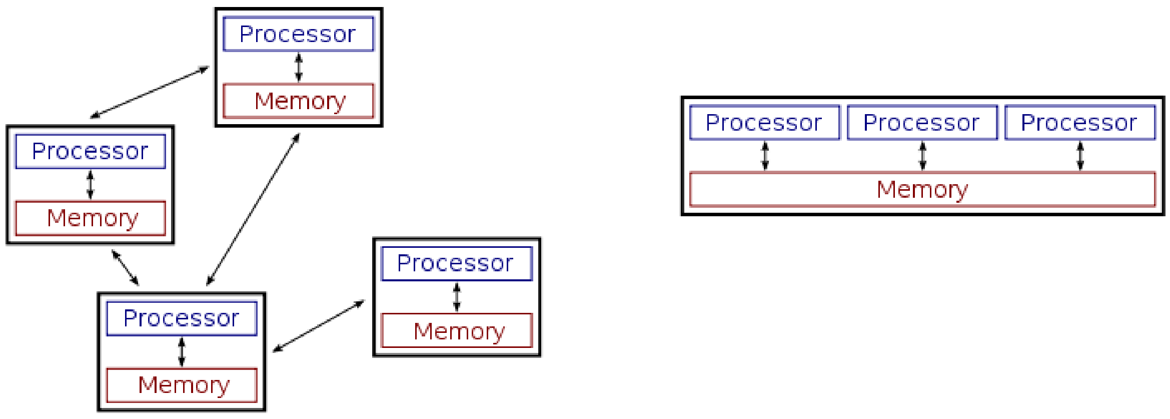


Figure 2.2 Distributed and Parallel Computing

The Distributed system on the left in figure 2.2 has memory located on each node and data can be transferred by communication links. The diagram on the right represents a parallel system where each processor shares one memory.

Parallelism has been employed for many years in HPC, or *supercomputing*, and with power consumption by computers becoming a concern, parallel architecture with some distributed features, mainly in the form of multi-core processors, became the typical form of a modern supercomputer. [11]

"Old [conventional wisdom]: Increasing clock frequency is the primary method of improving processor performance. New [conventional wisdom]: Increasing parallelism is the primary method of improving processor performance... Even representatives from Intel, a company generally associated with the 'higher clock-speed is better' position, warned that traditional approaches to maximizing performance through maximizing clock speed have been pushed to their limit." [11] Thus current supercomputers may have from a few hundred processors to more than a hundred thousand.

The fastest supercomputing systems in the world are ranked in the TOP 500 List of Supercomputing Sites [2] and are usually called high-end computing (HEC) systems or extreme-scale HEC systems because of the number of the processors they employ. The HEC systems are used to analyse and solve advanced computation problems in a vast number of scientific disciplines such as climate dynamics, nuclear astrophysics, fusion energy, nanotechnology, etc. The problems include various numerical simulations of already known events (earthquake, tsunamis) and unobserved situations (weather, sub-atomic particle behaviour), model fitting and data analysis (oil exploration geophysics, computational linguistics), as well as optimization of known scenarios [3].

Other types of Computing:

1. *Sequential Computing* is another way to do computations. As the name suggest to make computations one after another. This is not a goal of HPC.

2. *Scientific Computing is a* field of study to structure mathematical models, to use numerical solution techniques, to use computers to analyse and solve scientific problems [12].

3. *Cluster Computing* is the technique of linking two or more computers into a network, usually through a local area network (LAN), in order to take advantage of the parallel processing power of those computers.

4. *Grid Computing* is a form of networking which harnesses unused processing cycles of all computers in a network for solving problems too intensive for any stand-alone machine.

5. *Cloud Computing* a type of computing, comparable to grid computing that relies on sharing computing resources rather than having local servers or personal devices to handle applications. The aim of cloud computing is to apply traditional HPC power, to perform thousand of thousands of computations per second.

6. *Petascale Computing* – is a new term based on having petaflops in architecture. Previous generations were called M*ega, Giga, Tera* using teraflops and the next generations are called: *Exa, Zetta, Yotta, Xona, Weka, Vunda, Uda, Treda, Sorta, Rinta, Quexa, Pepta, Ocha, Nena, Minga, Luma* and so on [73].

The challenge of Petaflops supercomputers is to solve some of the most difficult scientific problems of national and global importance in areas such as combustion and fusion, climate modelling and making models in molecular chemistry and other areas. Annually these resources are opened to scientists, engineers and researchers from universities, industry, government, laboratories and non-profit organizations [3].

2.1.3 Applications Projects

The HPC system is a powerful and important tool, able to open horizons for computer scientists in making computations, understanding scientific problems, making visualizations of climate dynamics and multimedia cartoons, making models in molecular chemistry, biology, fusion energy, looking inside the human body and much more (astrophysics, physics, computer science, fusion, engineering) without making physical objects in the real world [4]. These innovations make the wheel of science spin with powerful and productive results.

In 2009, ORNL allocated the following projects on its HPC systems [3]:

Astrophysics projects:

- Multidimensional Simulations of Core Collapse Supernovae;
- Numerical Relativity Simulations of Binary Black Holes and Gravitational Radiation.

Biology projects:

- Cellulosic Ethanol: Physical Basis of Recalcitrance to Hydrolysis of Lignocellulosic Biomass;
- Gating Mechanism of Membrane Proteins.

Chemistry projects:

- Molecular Simulation of Complex Chemical Systems;
- An Integrated Approach to the Rational Design of Chemical Catalysts.

Climate projects:

- The Role of Eddies in the Meridional Overturning Circulation;
- Eulerian and Lagrangian Studies of Turbulent Transport in the Global Ocean.

Computer science project:

- Performance Evaluation and Analysis Consortium End Station.

Engineering project:

- High-Fidelity Simulations for Clean and Efficient Combustion of Alternative Fuels.

Fusion projects:

- Fluctuation Spectra and Anomalous Heating in Magnetized Plasma Turbulence;
- High-Power Electromagnetic Wave Heating in the International Thermonuclear Engineering Reactor (ITER) Burning Plasma.

Physics projects:

- Computational Nuclear Structur;
- Modeling Heliospheric Phenomena with an Adaptive, (Magneto Hydrodynamics) MHD-Boltzmann Code.

The allocated time for each of these projects can reach 60 million hours, but it may take much longer time if a failure occurs.

2.1.4 Failures and Faults

A Failure is an abortion, i.e. the inability of a system to perform its required functions within specified performance requirements. Failures are either random, in hardware or systematic, in hardware or software. A failure should not be confused with a fault.

A Fault is a defect, i.e. an abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function. It is possible for a machine or a software to have faults without necessarily failing.

Research [13] shows that failure data can be categorised into 6 categories: human error, environment, network failure, software failure, hardware failure and unknown failures. For example, a hardware failure is a fan failure that causes a computer to overheat. Another example of the hardware failure is a hard disk drive (HDD): XTORC is a cluster at ORNL. It is used to test some applications before they run on HPC systems. XTORC has 62 nodes, but currently 48 are in use. This is due to faulty architecture. These failures occur because these computers are not precisely designed for heavy computations.

The failures are the result of faults of architecture issue, human professionalism, coding and others. Comparison was done between failures occurring during 9 years from 22 HPC systems, including 4750 machines and 24101 processors at LANL [13]. The data that were looked at are the time when the failure started, the time when it was resolved, the system, the node affected and some other values [13].

Each system gave number of different results about how many failures were encountered on each system, the failures are dramatically differs from one to another, and each system gave different percentage of failures from the all failures [13].

The result shows that the main source of failures is hardware; the next largest contribution was from software and the last that are important to consider are unknown failures [13]. Also close attention was paid to systems memory and parallel file systems [13].

It is difficult to see the whole picture of failure problems from the averaged number of failures and to make judgement on faults because each case is unique if the systems are different. Even on the same system, failures constantly change during the year. Also important to node that even if a system looks identical, nodes might differ in number of processors and network interfaces [13].

There can be different causes for failures in HPC systems [5]:

- Programming errors, some discrepancies between implementation, specification, and use cases of a software. These design errors do not make the system perform in expected behaviour.
- System overloading can decrease system performance and cause a failure because the system will exceed its resources. An example is a denial-of-service attack.
- Wearing down cause. For example: power supply, processor, cooling fan, and disk failures.
- Pre-emptive or protective measures, for example if a system experiences unusually high temperature readings, the system will shut down in order to avoid damage.
- Other causes for failures exist, such as weather conditions, humans incorrect use of a system.

There is a difference between hardware failures and software failures. A hardware failure is a failure of a hardware component that needs to be replaced, for example a failed chip. A software failure is a failure of a software caused by having errors in a programming code.

The term *outage* or *downtime* is used to describe any kind of deviation from specified system behaviour, whether it is expected or unexpected. All faults and failures can be categorised as unplanned downtime. And planned downtime is performed to improve system functionality, such as to perform maintenance operations or software upgrades [5].

2.1.5 Fault Tolerance

Fault Tolerance (FT) is a mechanism or approach to handling failures. There are many existing approaches and some of them are mentioned below:

- *Reactive FT* is based on recovery of an application from experienced failures.
- *Algorithmic-based FT* detects and corrects or ignores permanent and transient errors in matrix operations on systolic arrays.
- *Practical Byzantine FT* is a sub-field of error handling research inspired by the *Byzantine Generals' Problem* [86].
- *Proactive FT* is based on avoiding failures in applications by using migration.
- *Holistic FT* (or *Hybrid FT*) is a combination of Reactive FT and Proactive FT.

2.2 Previous Work in Proactive Fault Tolerance

HPC systems are increasing in size from thousands of processors to hundreds of thousands of processors and beyond. Thus failures are likely to be more frequent than they were before. Therefore a lot of effort should be put into schemes for dealing with failures.

A new approach is proposed to deal with failures in a proactive way instead of waiting for failures to occur and react to these failures. The proactive approach requires a failure to be predictable. After a failure is predicted a decision is made to do a migration from a deteriorating node to a spare node [61].

A new approach to handling failures on HPC systems is called Proactive FT [22]. Proactive FT deals with failures by avoiding them and keeping the applications alive. Proactive FT proposes greater fault resilience against predictable failures than reactive FT. In contrast, Reactive FT deals with failures by recovering from them. These two approaches complement each other to have better resilience.

Proactive FT is a new technology that prevents failures from occurring during parallel computations by pre-emptively migrating applications or their parts away from the deteriorating node [22]. Monitoring techniques are used to collect metric values and compare these with their thresholds in order to not exceed this threshold so that if a metric value is reaching its threshold the migration process takes place. The idea with running application through migration will increase application mean-time to failure (AMTTF) above system mean-time to failure (SMTTF).

Proactive FT using migration is based on a *feedback-loop control mechanism* and consists of application health monitoring, analysis and reallocation [22].

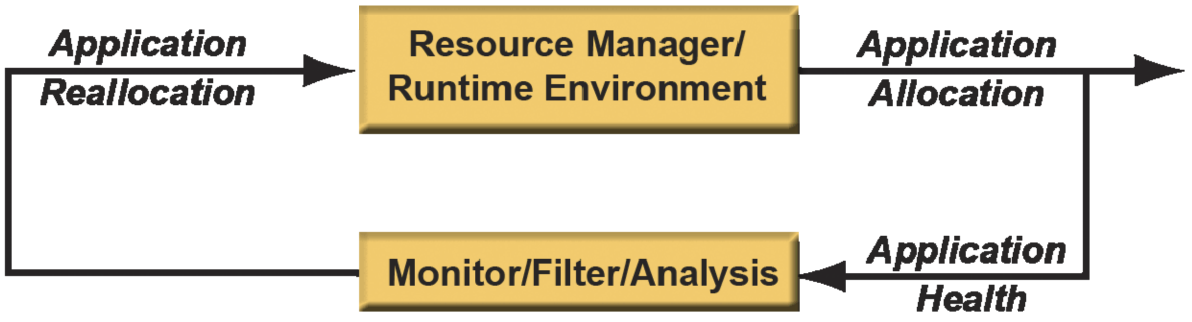


Figure 2.3 Feedback-Loop Control Mechanism [22]

The feedback loop mechanism shown in Figure 2.3 consists of two parts: monitoring system and migration system. This continuous process is based on accepting applications, allocating them on resources, monitoring resources and services, making analysis of the running processes and making predictions on failures. When a failure is predicted a migration is done either at application or at process level. This means that either a whole application or just part of an application is migrated away to another node or to another processor on the same node if there are not enough spare nodes allocated [22].

The core idea in Proactive FT [22] is to make a prediction. However some questions need to be answered such as is it possible to predict a failure and if yes how to do a prediction. An example will give some thoughts about making prediction. If a temperature of the system is constantly increasing it is likely that it will exceed its thresholds.

The proactive FT is based on (1) monitoring the metrics of a system, (2) decision making mechanism on predictions and (3) action to make before a failure occurs. For that the failures need to be analysed very carefully [22].

This Proactive FT has its own limitations:

- Not all failures can be predicted;
- Time - migration must be completed before failures occurs;
- Migration action only when it is necessary;
- Reliability analysis must be advanced to make correct predictions.

Unfortunately to use Proactive FT alone is risky. However, combination of reactive and proactive technologies provides more chances to have more efficient structures to handle predictable and unpredictable failures [22]. Also pause/unpause solution can be used.

Proactive FT in Message Passing Interface (MPI) applications via task migration [42] concentrates on using proactive approaches to migrate parallel applications from a processor to a processor when failures are imminent. It is assumed that the failures are predictable. The concept of processor virtualization and dynamic task migration are used to implement a mechanism that migrates tasks from one processor, that is about to fail, to a spare one. The approach consists of three major parts: (1) migrate Charm++ objects from an alarmed processor and point-to-point message delivery must function even after crash, (2) deal with a processor that is about to fail, (3) migrate Adaptive Message Passing Interface (AMPI) processes from an alarmed processor. These parts are interdependent.[42]

The paper [42] demonstrates the flexibility of the approach by providing performance data from experiments with existing MPI applications. The results show that proactive task migration is an effective solution to deal with faults in MPI applications. [42]

2.2.1 Reliability and Availability Analysis

As computers increase in scale and become more complex in the architecture, it is not surprising that failures occur more frequently. In this case a long running application on an HPC system will experience frequent interruptions and will be rolled back to start the computation all over again or from the last checkpoint [60].

One example from the LLNL showed that Mean Time Between Failures (MTBF) was about 160 days on a 512 node cluster [61]. But on BlueGene/L the same situation would end up with 17 node failures per hour. It is because of the availability issues. The solution was found to disable L1 cache on each node for applications running more than 4 hours [60].

The work [60] was done on exploring temporal and spatial correlations for failure predictions in coalition systems. For the temporal correlation a covariance model was developed with an adjustable time-scale. To analyse spatial correlation a stochastic model was used [60].

Probabilistic characterization to reduce runtime faults in HPC system [43] discussed to use hardware level monitoring, analysis and characterization to increase MTTI. This will increase application performance on HPC systems.

The paper [43] proposes to use a statistical approach in order to find anomalies in HPC systems. The work was done by (1) using OVIS tool [44] for hardware analysis and element characterization, (2) utilizing statistical characterizations in order to make correct resource management decisions, and future work is (3) decision making mechanism.

To predict software failures in HPC the software metrics need to be explored. For that a Bayesian Network (BN) model was build [51]. Software metrics can be predicted when they are fault-prone and when observations of complex relationships between these metrics are done [51].

Unfortunately, because of the security issues very little raw data on failures is available for public use. Therefore little analysis can be done on failure data to design a mechanism for prediction.

Some of the analysis was done [47] on collected failure data released to the public from LANL over 9 years [48]. The data has 23000 failure records from more than 20 different systems. The paper [47] concentrates on a statistical analysis of the data, analysing MTBF and Mean Time to Repair (MTTR), and finding major cause of failures.

The conclusion of this work [47] is that failure rates vary widely across systems and do not depend on the system hardware but on the system size, failures are roughly proportional to the number of processors on the system, and that there are more hardware failures than any other failures, i.e. software, network, human, unknown.

Analysing and visualising anomalies in HPC systems can be a first step to creating a Reliability Analysis. This work [77] consists of looking at raw numerical data aggregated at regular time intervals from a 48 node cluster. The analysing tools used included: R - an open source programming language and software environment for statistical computing and graphics, and GGobi - an open source visualization program for exploring high-dimensional data.

The aggregated data were clustered into groups. Careful attention was taken to having a small group or singletons of data values because these could show that some changes happened during computations and could be a fault or lead to a failure [77].

Furthermore investigations on the same data were done [77] using GGobi visualisation program. The program is able to find anomalies and automatically find attributes. Visualisation provides opportunities to see in different colours and in different dimensions [77]. Some experiments were done with injections of faults. When faults were injected manually into raw data, R and GGobi captured the anomalies in the clustered data.

The work in the paper [49] proposes fault-aware runtime strategies for spare node allocation and job scheduling on HPC. (*Job* is a unit of work for a computer, generally comprising an application program or group of related programs and the data, linkages, and instructions to the operating system needed for running the programs). This work together with failure prediction and FT techniques form a so called *Fault-Aware Runtime System* (FARS) for HPC. This FARS is capable of improving HPC systems performance if specific conditions are satisfied [49]. The FARS is still under development and some of the future work proposed includes: (1) collect workloads and failure events to improve FARS, (2) integrate FARS with fault-aware scheduling work, for example in [50] and (3) integrate FARS with failure prediction. FARS can be one of the solutions that will improve fault management in HPC.

2.2.2 Monitoring

System reliability and application resilience became very important when speaking about HPC system performance. As computer system components increase in size more performance problems are arising. The failures are increasing and it is important to improve FT. For that there should be a constant progress in handling failures. Monitoring framework are essential for this progress to happen. The paper [57] suggests 3 types of monitoring: node-level system monitoring, subsystem-level system monitoring and application-level system monitoring. Some of these monitoring approaches are complex, expensive, require specialized hardware, libraries but others are less expensive and less complex [57].

One of the important things to consider is accurate fault detections when speaking about resilient computing. Syslogs can provide some useful information about faults. And many HPC systems, if not all, use system logging mechanisms. Discovering new fault types can take unlimited amounts of time and knowledge of experts in this field [56].

The experiments were done [56] looking on syslogs of 512 node *Spirit'* Linux cluster. An algorithm was designed to observe that if similar computers in architecture execute similar work the results in the logs should be also similar to each other [56]. This approach can be used to improve resilience of HPC systems when a fault has occured or is imminent. Quick detection and response are important issues to consider [56].

While it is important to avoid failures whenever possible it is needed to be accepted that some of failures are inevitable and that progress needs to be done in spite of these failures. For example progress can be made by using failed node resources for other applications or for already queued jobs to resource managed [57].

2.2.3 Prototypes on Migration

Virtual Machine (VM) is an example of an operating system along with one or more applications running in an isolated partition within a computer. It enables different operating systems to run in the same computer at the same time as well as prevents applications from interfering with each other

Operating system-level virtualization is a method for splitting a single computer into multiple partitions called *virtual environments* (VEs), in order to prevent applications from interfering with each other. OS-level virtualization replicates components of the host operating system into each VE. The OS-level virtualization method differs from the traditional VM method, because it supports only the same operating system in each partition rather than different operating systems running simultaneously. Therefore, if the OS is Linux, all VEs must run Linux.

Xen hypervisor is an open source industry standard for virtualization on x86, x86_64, IA64, ARM, and other Central Processing Unit (CPU) architectures. It supports more than one guest OS [74].

An experiment was done [75] in designing VM-level pre-emptive migration for HPC using Xen. This system should provide proactive decision making and address load balancing. The overview of the system is shown below.

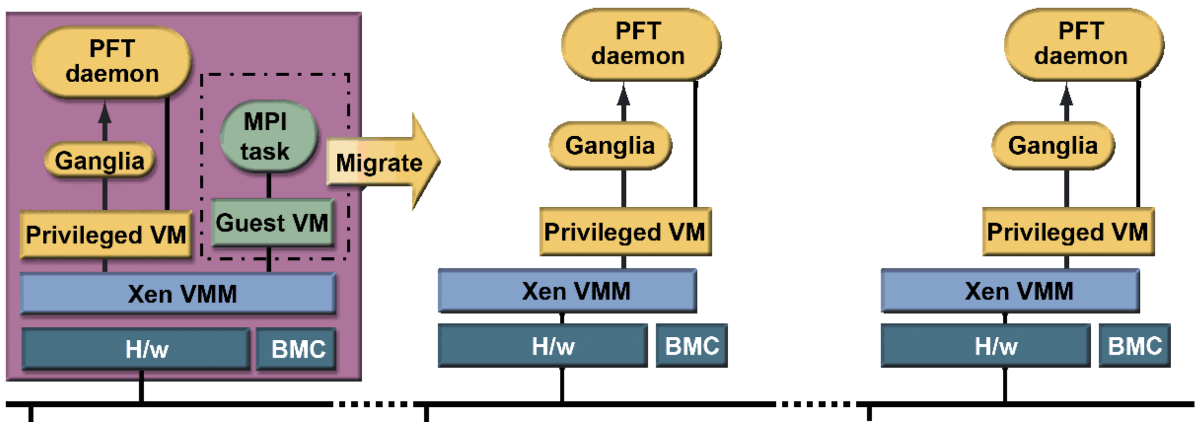
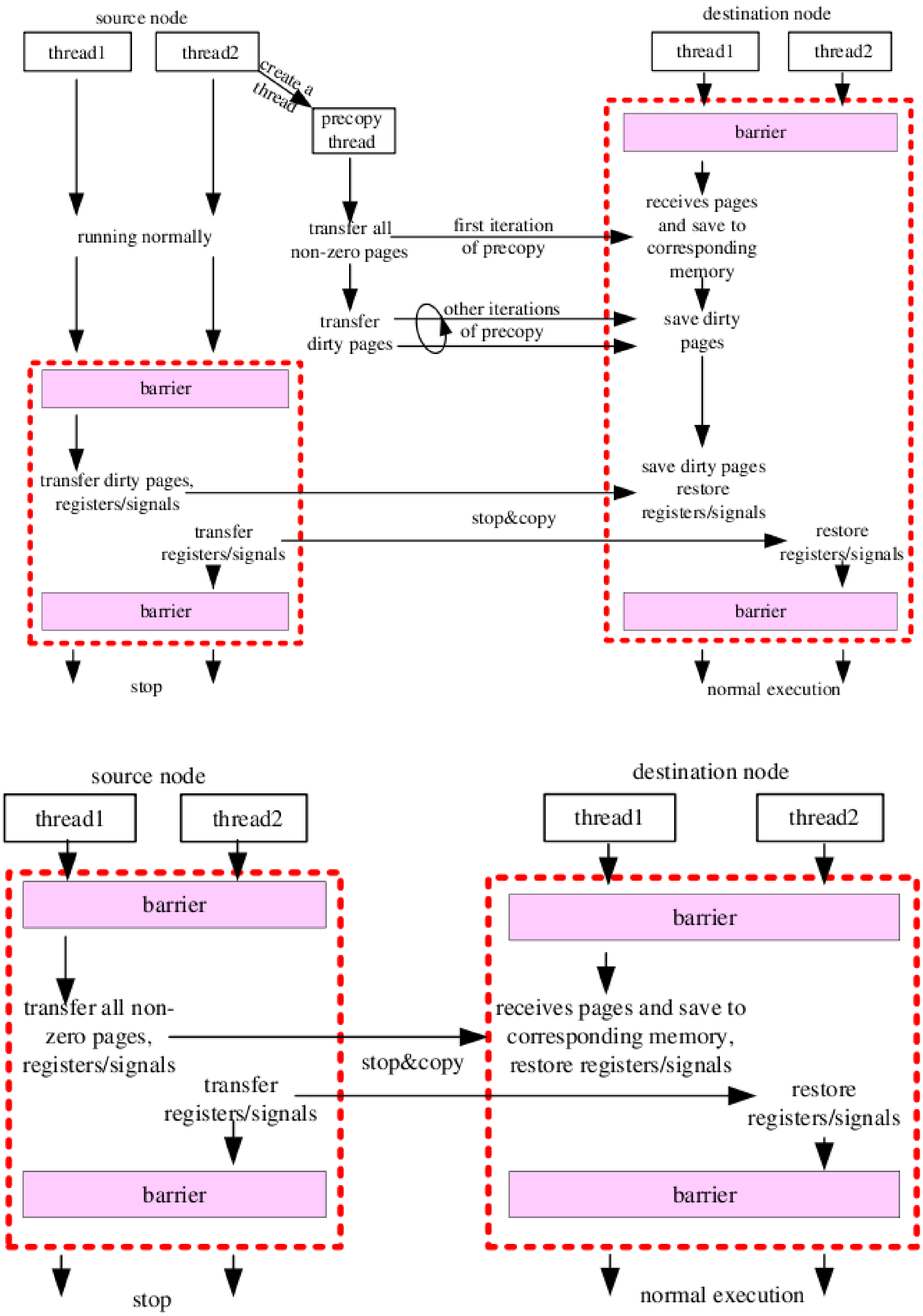


Figure 2.4 Overall Setup of the Components [75]

Xen Virtual Machine Monitor (VMM) handles the execution of all instruction on the virtual CPU and the imitation of all virtual devices. VMM is located on each node. On top of VMM runs Privileged VM and Guest VM. Privileged VM manages the Ganglia Monitoring System, which monitors the health of the system and Proactive Fault Tolerance daemon (PFT daemon), which initiates migration. Xen VMM also hosts a Guest VM, which make computations. When a node deteriorates the entire Guest VM moves to another node. First, Guest VM finds a spare node, then starts sending pages in chunks, then when everything is sent to the destination node Guest VM is stopped on the deteriorating node and is restarted on the spare node [75]. *Pages* is a block of program instructions or data stored in main or secondary storage.

Proactive live migration in HPC environments is described at process level and at the job level [76]. There are 6 steps to be followed in order to make live migration at the job level: (1) trigger a migration, (2) determine destination node, (3) transfer a memory snapshot of the process image to the spare node, (4) in-flight message coordination to reach a consistent global state, (5) migration process is stopped in order to send the last pages, and the last step is (6) recreation of the connection, restoration of messages and continuation to run jobs [76]. Steps 3-5 of process level migration are depicted in figure 2.5.



source node
thread1
thread2
create a thread
precopy thread
destination node
thread1
thread2
barrier
running normally
transfer all non-zero pages
first iteration of precopy
receives pages and save to corresponding memory
transfer dirty pages
other iterations of precopy
save dirty pages
barrier
transfer dirty pages, registers/signals
save dirty pages restore registers/signals
stop&copy
transfer registers/signals
restore registers/signals
barrier
barrier
stop
normal execution
source node
thread1
thread2
destination node
thread1
thread2
barrier
barrier
transfer all non-zero pages, registers/signals
receives pages and save to corresponding memory, restore registers/signals
stop&copy
transfer registers/signals
restore registers/signals
barrier
barrier
stop
normal execution

Figure 2.5 Process Level Migration (Kernel Mode in Dotted Frame) [76]

The picture on the top is a process level migration with memory pre-copy and the picture on the bottom is a process level migration without memory pre-copy. Process level migration is shown to be less expensive than migrating whole OS images under Xen virtualization. Proactive approach complements reactive by avoiding rollbacks if a failure is predicted. Thus, checkpoints may be cut in half if 70% of failures are handled proactively [76].

The failures that are encountered on computers can be hardware or software. If there is a hardware failure, like fan failure which leads to computer overheating, the migration can be predicted analysing the temperature thresholds that should not be reached. To look deeper into this problem extra supplies can be provided while migration is taking place, and after migration from the unhealthy node to a spare one the failure is left behind.

The situation is completely different with software failures such as errors in programming code. For example at the job level, when the failure is predicted by the VM and this VM is migrated with the application, it means that the failure will be migrated too and not avoided. In this case reactive fault tolerance such as checkpoint/restart is used.

2.3 Related Work on Fault Tolerance

Rollback recovery protocols can be checkpoint-based or log-based [53]. The checkpoint-based protocol is a solution to recover the HPC system when a failure occurs. The checkpoint-based rollback recovery is simpler and easier to implement than log-based, because there is no need to detect, log or replay non-deterministic events [53]. The protocol can be: (1) uncoordinated checkpointing i.e. a process can make a checkpoint whenever it is convenient, (2) coordinated checkpointing and (3) communication induced checkpointing.

The log-based protocol combines checkpointing with logging with non-deterministic events. The protocol can be pessimistic, optimistic or casual. The performance of a system will vary upon choosing different protocols. Some of these protocols do not have promising futures as the result shows, for example log-base recovery [53, 54].

2.3.1 Reactive Fault Tolerance

One of the techniques is *checkpoint/restart* and it is commonly used in HPC systems. A computer node state is copied to a shared stored memory at regular intervals. A shared stored memory on HPC systems is usually a high-performance networked file system. When a failure occurs during live computation the application is lost but can be recovered from the last saved state from the memory. Checkpoint/restart requires coordination of all nodes to have consistency. Checkpointing an application during computations is a pre-emptive method and is counted as planned downtime. Restarting an application is a reactive method and is counted as unplanned downtime. Mean time between repairs (MTBR) in checkpoint/restart systems is defined as the time to detect a failure, the time to rollback and the time to restart the computations from the last checkpoint.

One of the methods to checkpoint/restart can be done inside a Linux Operating System kernel. The method is called Berkeley Laboratory Checkpoint Restart (BLCR). This method is less portable. It cannot checkpoint and/or restore open sockets or inter-process communication objects, for example pipes or shared memory [62].

Local Area Multicomputer (LAM) [63] is an implementation of the Message Passing Interface (MPI) [64] standard used by applications for low-latency/high-bandwidth communication between compute nodes in HPC systems. LAM and BLCR work together to allow transparent checkpoint/restart of MPI-base applications.

For example, Transparent Incremental Checkpointing at Kernel-level (TICK) [65] provides incremental checkpointing support transparently to applications. The idea is to take a snapshot and reduce checkpoint data to a minimum by saving only the difference between the previous checkpoint and the last snapshot.

Another method is to use *Diskless Checkpointing*. Diskless Checkpointing is a technique for checkpointing the state of a long-running computation on a system without using stable storage. The checkpoint is stored in a memory and not on a disk. The advantage of storing in a memory is a faster process. Diskless Checkpointing provides high-performance and reliable storage on large systems [66].

2.3.2 Message Logging protocol

The approach is based on capturing all messages of HPC application including warning and error messages. Upon a failure the process is rolled back to a previous state. Checkpoint/restart mechanism can be combined with message logging in order to avoid rollback of an application from the very start. The solution is MPICH-V [70].

MPICH-V is a research effort [70] with theoretical studies, experimental evaluations and pragmatic implementations attempting to provide an MPI implementation based on MPICH, featuring multiple FT protocols. MPICH-V provides a automatic FT MPI library, i.e. a totally unchanged application linked with the *mpich-v* library is a FT application. MPICH-V is used for a) large clusters, b) clusters made from collection of nodes in an LAN environment, c) Grid deployments harnessing several clusters and d) university campus and industry wide desktop Grids with nodes, i.e. all infrastructures featuring synchronous networks or controllable area networks.

There are many MPICH-V protocols [70]:

- MPICH-PCL has a new communication channel, which is called *ft-sock* and based on the TCP sock channel, and two components, a checkpoint server and a specific dispatcher, supporting large scale and heterogeneous applications. It also has migration developed capability. Computation is able to restart from a given checkpoint wave.

- MPICH-VCL is designed for extra low latency dependent applications. No overhead during fault free execution because of Chandy Lamport Algorithm. But there is a disadvantage: all nodes require restart in the case of a single fault. As a result, it is less fault resilient than message logging protocols, and is only suited for medium scale clusters.

- MPICH-V1 is another fault tolerant protocol designed for very large scale computing using heterogeneous networks. This protocol is used for Desktop Grids and Global computing because it can support a very high number of faults, but requires a larger bandwidth for stable components to reach good performance.

- MPICH-V2 is designed for very large scale computing using homogeneous networks, usually clusters. MPICH-V2 requires a small number of stable components to achieve good performance on a cluster.

- MPICH-VCausal is designed for low latency dependent applications which must be resilient to a high fault frequency. It includes the advantages of the other message logging protocols with direct communication and absence of acknowledgements. Thus with these advantages MPICH-VCausal protocol can provide computation progress even with high fault frequency and can avoid high latency impact.

2.3.3 Algorithm-Based Fault Tolerance

Algorithm-based fault tolerance is a low-cost FT scheme to detect and correct permanent and transient errors in certain matrix operations on systolic arrays or to ignore failures and still produce acceptable results. The key goal is to encode the data at a higher level using *chekcsum* schemes and redesign algorithms to operate on the encoded data. The contribution to original development is to create algorithms for which algorithm-based FT can be used at the same time. A new focus is to obtain an efficient FT matrix-matrix multiplication *suroudtine* [67].

FT Message Passing Interface (FT-MPI) is a full 1.2 MPI specification implementation that provides process level FT at the MPI API level. FT-MPI is built upon the fault tolerant HARNESS runtime system. FT-MPI survives the crash of n-1 processes in an n-process application, and, if required, can restart them. However, it is still the responsibility of the application to recover the data-structures and the data on the failed processes [68].

Open Message Passing Interface (OpenMPI) project is an open source MPI-2 implementation that is developed and maintained by academics, researchers, and industry partners. OpenMPI is able to combine the expertise, technologies, and resources from all across the HPC community in order to build the best MPI library available. OpenMPI provides advantages for system and software vendors, application developers and computer science researchers [69].

2.3.4 Hybrid Fault Tolerance

Hybrid FT is a proposed [71] solution of combining Reactive FT and Proactive FT in a model to use it in the stateful firewall semantics to increase the overall performance of cluster-based fault-tolerance stafeful firewalls. The reactive fault-handling approach is not efficient anymore in HPC but using only proactive fault-avoidance approach is too risky, because (1) not all faults can be predicted, (2) there might not be enough time for completing a migration. The research shows [71] that Hybrid FT reduces the computational resource consumption. Using a low barrier in Naive Bayes to classify states the Hybrid solution ensures that the reduction in the state replication is 40-50% and wrong predictions are around 2%. Future work is concentrated on evaluation of sophisticated machine-learning approaches [71].

2.4 Summary

The are many different ways of handling failures in HPC systems. FT can be reactive, algorithm-based, practical Byzantine, proactive, holistic or hybrid. As the HPC system has increased in size, the most commonly used reactive FT is no longer capable of handling failures in the same efficient way. Proactive approach is able to deal with failures by avoiding them. Results and studies show that it is an acceptable way of dealing with failures when they are predictable.

3. PRELIMINARY SYSTEM DESIGN

“Making a system reliable is not really hard, if you know how to go about it. But retrofitting reliability to an existing design is very difficult”. (Butler W. Lampson)

This chapter examines importance of the reliability, availability and serviceability of a HPC system. It discusses the requirements and specifications of the existing RAS Framework for handling failures. The chapter is concentrated on a RAS Framework Engine Prototype as a migration mechanism.

3.1 Reliability, Availability and Serviceability of a System

Reliability, *Availability* and *Serviceability* (RAS) is a set of important attributes to be taken in to consideration when designing, implementing, manufacturing, purchasing or using a computer product or a part of it. RAS attributes are needed to define specifications for computer systems and originally were considered and applied only to hardware. But later on [20] RAS became relevant both to software and hardware and was applied to other relevant objects such as networks, application programs, operating systems (OS), personal computers (PC), servers and supercomputers. Each attribute needs to be analysed.

The term *reliability* refers to the ability of a computer-related hardware or software component to maintain consistently the desired performance according to its specifications. Theoretically speaking it is possible to implement a reliable product that has no errors. But reality shows that this is difficult to achieve by seeing new updated version coming on the market after the first release. In other words the trick is not to design a reliable product, in a way this is quite easy, but to improve reliability of a system, that already exists, which can be very difficult. That is why it is common for vendors to express product reliability as a percentage.

Reliability requires two conditions to be fulfilled: failure avoidance and robustness. A failure occurs when a system is not able to perform its required functions. The causes are faults, i.e. mistakes that are nothing else than errors that are left uncorrected. Failures can occur during installation because, for instance, a switch backwards. Another example, a failure may appear after incorrect usage of metric values when inches are represented in centimetres. [21]

Causes for failures in HPC system [4]:

- Design errors, such as programming errors, discrepancies between implementation, specification and other.
- System overloading can decrease system performance and cause a failure because the system will exceed its resources.
- Wearing down, such as power supply, processor, cooling system, disk failures and other.
- Pre-emptive or protective measures, such as if a system experiences unusually high temperature readings, thus the system will shut down in order to avoid damage.
- Other causes for failures exist, such as human carelessness, nature causes or other accidents.

Reliability is improved by avoiding these failures with a combination of knowledge-based engineering and the problem-solving process [21].

Robustness is the ability of a system to function smoothly, in other words to avoid failures under a specified range of conditions that are experienced in the field. Robustness improvements are made by testing systems under different conditions. It can be a big challenge to improve robustness because there are a broad set of experiences: environmental and operating conditions. Another challenge is effective system engineering, but this one is not the most important [21]. Failure-mode avoidance is an approach to make changes in system design, but this is a theoretical approach because making changes will not always be effective [21].

Reliability is a probability of failure to occur under specified operating conditions. The probabilistic concepts are: survival function, failure rates and mean time between failures (MTBF):

- *Survival function or reliability function* is a property of any random variable that maps a set of events, usually associated with mortality or failure of some system, onto time. It captures the probability that the system will survive beyond a specified time.
- *Failure rate* is the frequency at which a component fails.

Figure 3.1 shows the difference between time to repair, time to failure and time between failures.

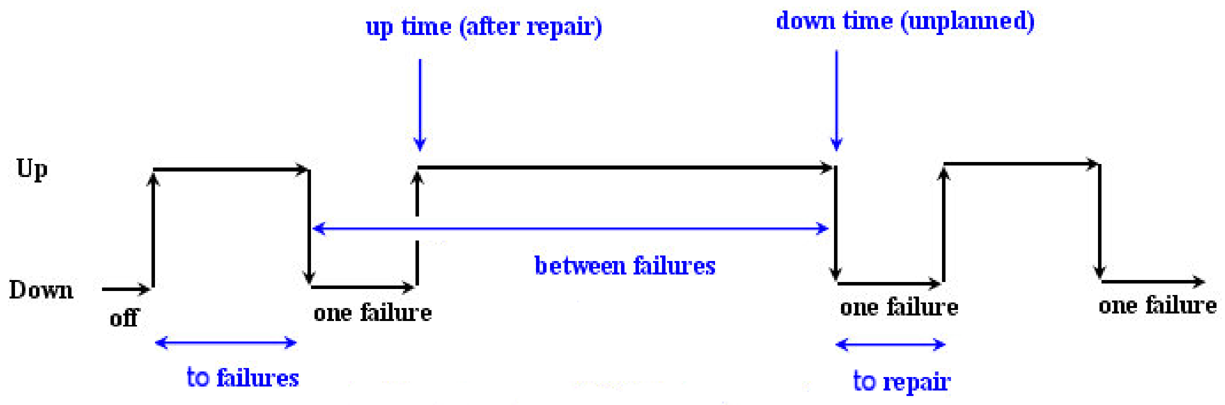


Figure 3.1 Time to Failure, to Repair, Between Failures

Uptime is time during which a system is working without a failure. *Downtime* is time during which a computer is stopped functioning due to hardware, operating system or application program failure. Thus, MTBF is the average time a device can function before failing. Another term to consider is MTTR. MTTR is the time taken to repair a failed hardware module.

Mean time to repair (MTTR) is the average time between uptime and downtime. *Mean time to failure* (MTTF) is the average time between downtime and uptime. *Mean time between failures* (MTBR) is the average time between failures.

Availability is the ratio of time a system or component is functional to the total time it is required or expected to function, i.e. MTTF divided by MTBF. This is usually expressed as a direct proportion (for example 9/10 or 0.9), but sometimes it is also shown as a percentage. It is normally expressed in terms of average downtime per week, month or year, or as total downtime for a specific week, month, or year [20]. In another words measured availability is a percentage of time that a system is available.

System availability can be seen in the Table 3.1 [4]. The higher system availability causes lower annual downtime:

9s	Availability	Annual Downtime
1	90%	36 days and 12 hours
2	99%	87 hours and 36 seconds
3	99.9%	8 hours and 45.6 seconds
4	99.99%	52 minutes and 33.6 seconds
5	99.999%	5 minutes and 15.4 seconds
6	99.9999%	31.5 seconds

Table 3.1 System Availability [4]

The availability of the system directly depends on the availability of each component of the system. And each component can depend on each other. The higher the components dependability on each other the lower is the availability of the system. But in parallel computing, the more redundant components the system has the more availability is offered on the system [4].

There are 3 types of availability that a system can have [78]:

- *Basic Availability*, when a system is designed, implemented and deployed with hardware, software and procedures components just to satisfy its basic functional requirements. The system will give the right results and function properly.

- *High Availability*, when a system has Basic Availability and has enough sufficient redundancy in hardware, software and procedures components in order to mask certain defined faults. To be clear further explanations are given for the terms sufficient, mask and certain.

  Faults are defined as an unexpected deviation from specified behaviour. To *mask* a fault means to use a shield to make a fault unobservable, i.e. no deviations from specified behaviour occur. *High Availability* is a degree of *transparency* in which variations of systems occur. The degree of transparency can be: (1) Manual Masking, (2) Cold Standby, (3) Warm standby and (4) Hot Standby, also called *Active Replication*.

  Sufficient means to satisfy hardware, software and procedures requirements, i.e. guarantee to have a specific quality of service. Thus sufficient redundancy implies that hardware, software and procedure faults must all be masked.

  Certain implies that not all faults can be masked. Because there is a need to restrict the number of faults that will be compensated in a Highly Available system.

- *Continuous Availability*, when a system has High Availability and applies High Availability also to planned downtimes. To be more precise High Availability compensated for unplanned downtime and Continues Availability has a masking strategy that deals with unplanned and planned downtimes.

Serviceability is an expression of the ease with which a component, device or system can be maintained and repaired. Early detection of potential problems is critical in this respect, but it can be achieved by using proactive FT approach, i.e. by analysing already known failures and then making predictions and then migrating away from expected failures [22].

There are some limited abilities to correct problems automatically before they actually occur. The example of corrections could be built-in features of OS such as Microsoft Windows XP, auto-protect-enabled anti-virus software and spyware detection, some removal programs. The desired goal is to have the shortest downtime possible in maintenance and repair operations [23].

Some of the key elements of RAS on the system are [23]:

- Over-engineering;
- Duplication - gives extensive use of redundant systems and components;
- Recoverability - use of reactive FT method for example;
- Automatic updating - keeping OS and applications updated without user intervention;
- Data backup - helps to restore lost data in an event of loss of critical information;
- Data archiving - needed to minimise used space and keep necessary records of data in case of recovery needs;
- Power-on replacement, a solution to hot swap components or peripherals;
- Use of virtual machine - eliminates the impact of OS or software faults;
- Continuous power, it can perform an interruptible power supply in case when switching from one power to another;
- Backup power sources, for example batteries and generators, it will keep systems operational during long interruptions in commercial power.

3.2 RAS Framework

One of the first ideas for improving the fault tolerance mechanism came up in 2007 [24]. The failures were increasing in numbers due to a well-known reason: the number of components in supercomputers was increasing and *Reactive approach* could not deal with so many of failures any more in the same efficient way as it used to before when used in a less advanced system.

There is no a single perfect solution to have high-level RAS for all HPC systems. A proposed approach [24] describes mechanisms suitable to provide a wide range of fault handling solutions. Each of these mechanisms can be used individually or used with other mechanisms in order to have a better solution for a specific HPC system. The core of RAS mechanisms is a proactive fault handling technique for high scale HPC systems. The fault handling solutions are prediction, detection, recovery and avoidance. These mechanisms form a RAS Framework, in figure 3.2.

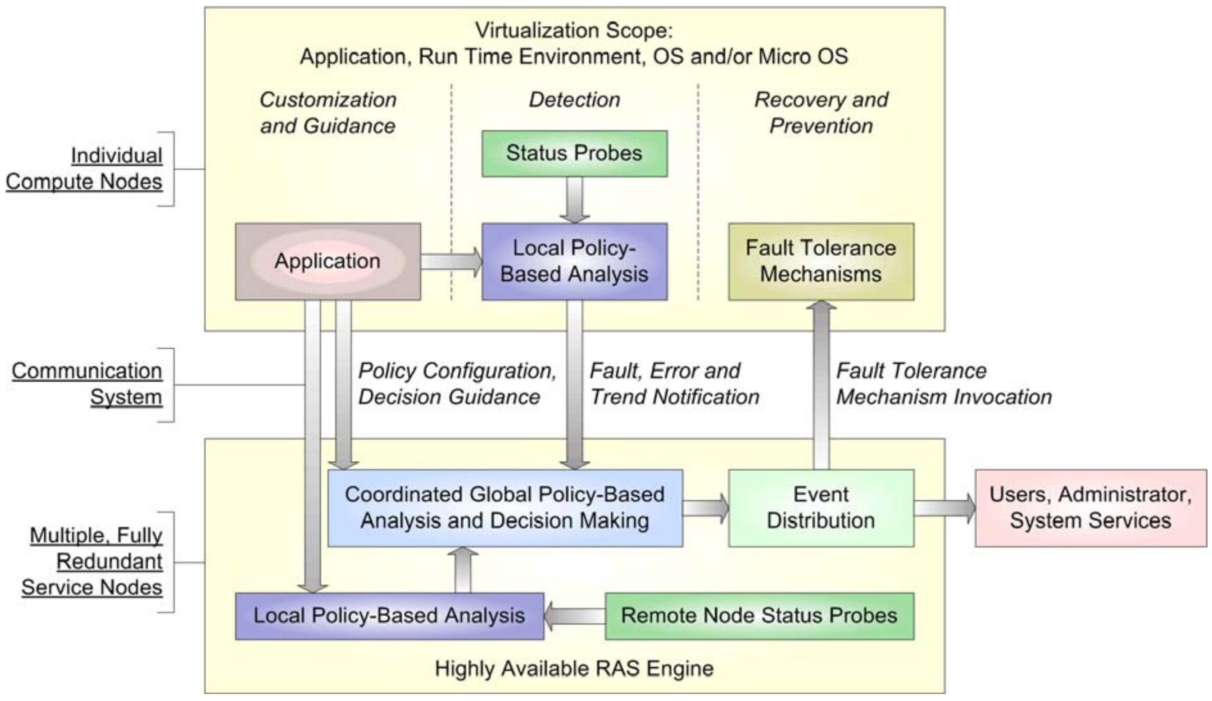

Virtualization Scope:
Application, Run Time Environment, OS and/or Micro OS
Customization and Guidance
Detection
Recovery and Prevention
Status Probes
Individual Compute Nodes
Application
Local Policy-Based Analysis
Fault Tolerance Mechanisms
Communication System
Policy Configuration, Decision Guidance
Fault, Error and Trend Notification
Fault Tolerance Mechanism Invocation
Coordinated Global Policy-Based Analysis and Decision Making
Event Distribution
Users, Administrator, System Services
Multiple, Fully Redundant Service Nodes
Local Policy-Based Analysis
Remote Node Status Probes
Highly Available RAS Engine

Figure 3.2 RAS Framework [30]

Each of the RAS Framework components can be replaced by open source solutions or commercial products. A core, a highly available RAS engine, is running on all the nodes of the HPC system and collects monitoring data from hardware and software, then uses an analysing tool to filter the events, and a decision making tool for making predictions and trigger a migration. After a prediction is made for a failure, proactive fault tolerance mechanism reallocated a running application from deteriorating node to a spare node by using pre-emptive migration. The Analysing tool uses monitoring data online and also uses stored historical data offline to generate a statistical model for making predictions and uses fault tolerance mechanism in order to avoid failures [24].

Proactive fault avoidance mechanisms aim to improve the efficiency of existing reactive fault recovery solutions. Proactive measures avoid failures in contrast to reactive that experience failures.

For example, an Analysing tool that looks at monitored data can indicate that the temperature is increasing and will go above its threshold. In this case the tool can use the proactive measures to migrate a process, task or VM away from a compute node that is about to fail [24].

The proposed RAS Framework can be tuned to use reactive fault recovery together with proactive fault avoidance. This solution will give better failures resilience on HPC systems and checkpoint frequencies can be minimized. Recovery from a failure is used in the case of unpredictable failures [24].

Therefore the preliminary system design of the RAS Framework [24] is based on a combination of the existing approach of Reactive FT with a new feature of Proactive FT in order to explore and integrate RAS Framework on HPC system to deal with failures. The goal is to have major key elements of high-level RAS for extreme-scale HPC systems [24]:

- Online *reliability analysis* using historical and system health monitoring data for identifying predictable failures;
- *Proactive fault tolerance* mechanism using pre-emptive migration, to move running applications away from the nodes that are about to fail;
- Enhanced *reactive fault tolerance* technology to adapt actual and predicted system health threats; and
- *Holistic fault tolerance* through a combination of proactive and reactive fault tolerance technologies.
- Other features can be added during integration, testing and implementation processes, such as, for instance, data archiving.

3.3 RAS Framework Engine Prototype

The aim of this thesis it to fulfil the requirement of the RAS Framework by implementing its core part – the RAS Framework Engine Prototype based on Proactive FT only. The figure below shows the desirable result using Proactive FT which is compared with Reactive FT.

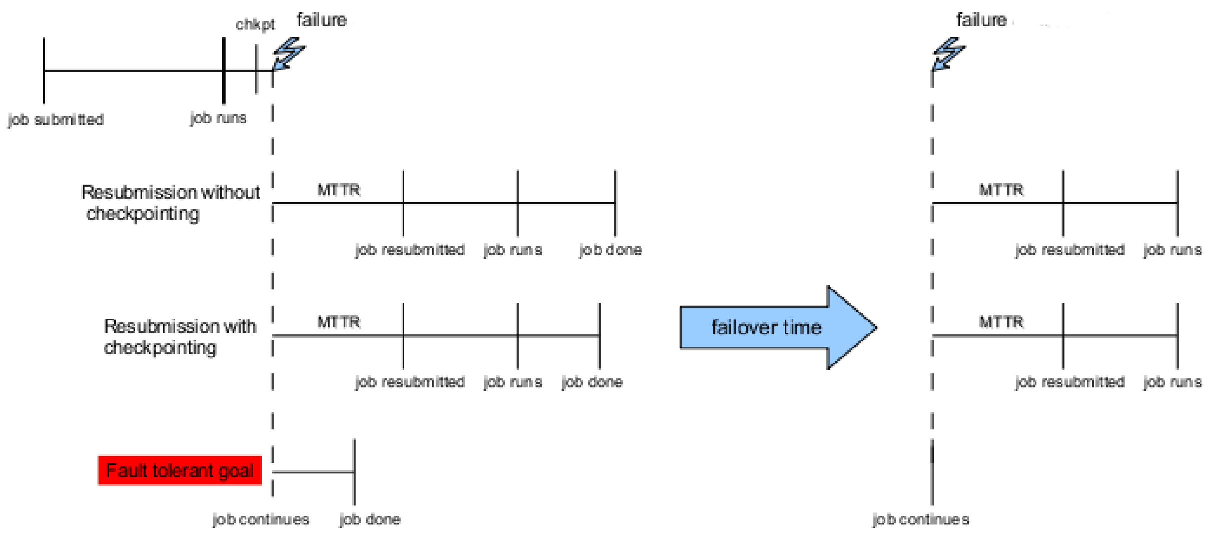


Figure 3.3 Comparison Between Proactive and Reactive Failures Handling [31].

Proactive FT is based on a feedback loop control mechanism (chapter 2). There are 4 types of feedback loops. The first type is the basic one. Each of the following types improves the previous one by an additional feature [22]. The following paragraphs of this section describe the types in details.

Types 1-4 have some or all of the following components [22]:

- *System logging*;
- *Job and resource monitoring/management;*
- *Environmental system monitoring*;
- *Reliability and availability analysis* to be used online, at present time and reliability and availability analysis to be used offline using historical data stored in a database;
- *Database*;
- *Migration* mechanism enabling avoidance of most of the failures rather than experiencing them.

Proactive FT mechanism use pre-emptive migration [22] in order to migrate from one compute node to a spare one, when a failure is imminent, before the actual failure occurs. In this case the failures can be avoided and not experienced, compared to Reactive FT approach.

Component 1: System Logging.

The system log file contains events that are logged by the OS components. These events are determined in advanced by the OS itself. By reading log messages, users can get information about system usage by other users, software, about level of a message, for example warning, error or information, device changes, device drivers, system changes, operations. Other information is provided about time used and so on. System logging need to fulfil requirements for the RAS Framework: scalability, widely supported, easy to use, easy to interface with database. Typical system logging are system logging daemon (syslogd), kernel logging daemon (klogd) and more advanced system logging is syslog-ng. System logging messages can be analysed online by Reliability Analysis or stored directly to a database.

Component 2: Job and Resource Monitoring

A Job scheduler (JS) and a resource manager (RM) take care of allocating submitted jobs in parallel computing on compute nodes. Usually the JS and the RM work as one, but more advanced software have one of them taking care of finding resources and the other one submitting applications to these resources.

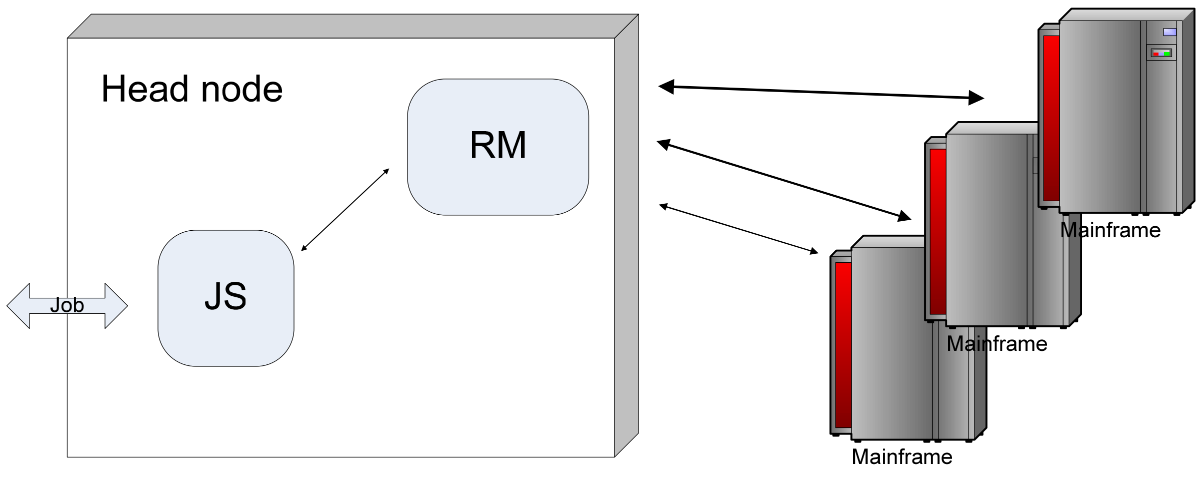


Figure 3.4 Interaction Between HPC System Job Scheduler and Resource Manager

The RM provides control over batch jobs and distributed compute nodes. The RM communicates with all compute nodes on the system and monitors memory usage and a processor utilization for all the nodes. The JS software schedules, prioritizes and allocates the given tasks on the resources. The JS gives administrators extensive control over the system. Generally speaking the JS tells what to do with tasks on the system and the RM does it. The JS and the RM need to fulfil the requirements for the RAS Framework: scalability, widely supported, easy to use, easy to interface with database, migration support. Most known JS and RM software are: Moab Cluster Suite, Maui Cluster Scheduler, Torque resource Manager. Information about submitted jobs can be analysed online by Reliability Analysis or stored directly to a database.

Component 3: Environmental System Monitoring.

Any computational resource can suffer from performance problems. Problems can reduce performance or can cause services to fail. Monitoring can help to understand what triggered a problem and can prevent it in the future [25]. System monitoring provides some knowledge about the state of a service. The *knowledge* is a result of a process pasted through 3 components: *monitoring target*, *monitoring/collecting agent* and *data storage*, shown in Figure 3.5.

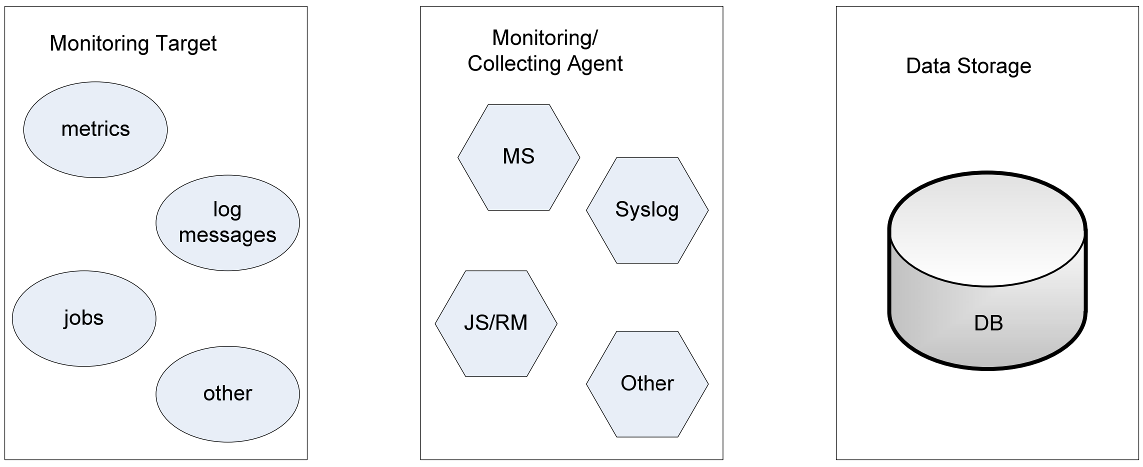


Figure 3.5 Three-Component Model for Data Monitoring

A Monitoring System (MS) takes responsibility to monitor many different computer components or service software which is in use and collect the values of the metrics. Example of computer components are: a fan, a network switch, an air-conditioning unit. Example of a system service is performance metrics. Data storage is discussed later on in this chapter. The MS need to fulfil the requirements for the RAS Framework: scalability, widely supported, easy to use, easy to interface with database. Some MS examples are: Ganglia Monitoring System, MonAMI Framework, OVIS tool, Nagios Network Monitoring System. Monitoring target values can be analysed online by Reliability Analysis or stored directly to a database.

The first type of feedback loop control mechanism [22] to be mentioned in designing a RAS Framework Engine Prototype consists of Resource Manager, Runtime Environment and Monitoring System, shown in Figure 3.6. *Runtime Environment* (RE) provides services for programs to be executed simultaneously on different compute nodes in a way that these nodes can communicate with each other in parallel.

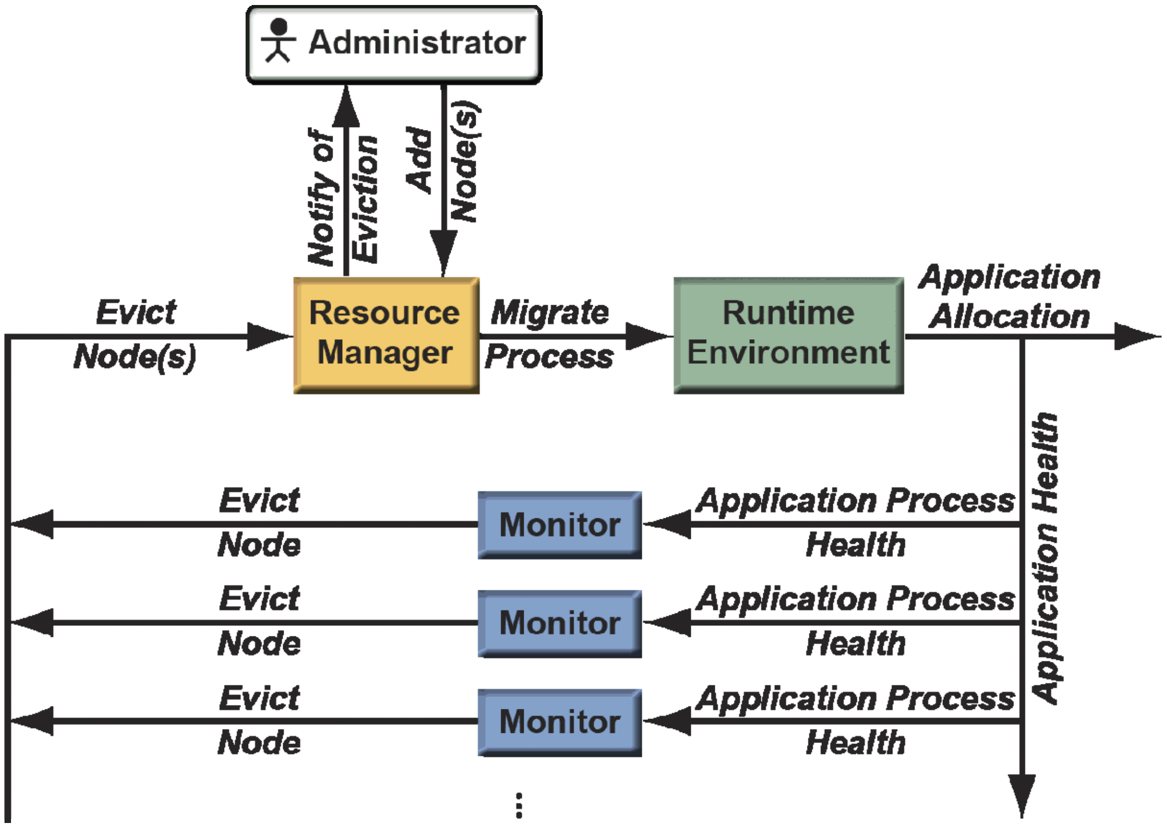


Figure 3.6 Type 1 Feedback-Loop Control Mechanism [22]

This feedback-loop control mechanism consist of the JS/RM, the RE and the MS. Each compute node has a MS which helps to observe progress of the system and application health parameters all the time. The MS will notify the RM on threshold exceeding fault appearance. Then the RM makes a decision of reallocating the application by migrating from the node in use to a spare node. This type of migration is not very efficient because it can happen that the application will migrate when it is not really needed or may not migrate quickly enough, for example when fan fault occurs and it leads to automated shut down of a system. Type 1 has been tested with OpenIMPI and Ganglia [27, 28]. OpenIPMI can be used with any feedback-loop control mechanisms [22].

The type 2 feedback-loop control mechanism [22] is shown in Figure 3.7. This includes a filter to selectively analyse data. The filter can be a trigger mechanism or machine learning mechanism.

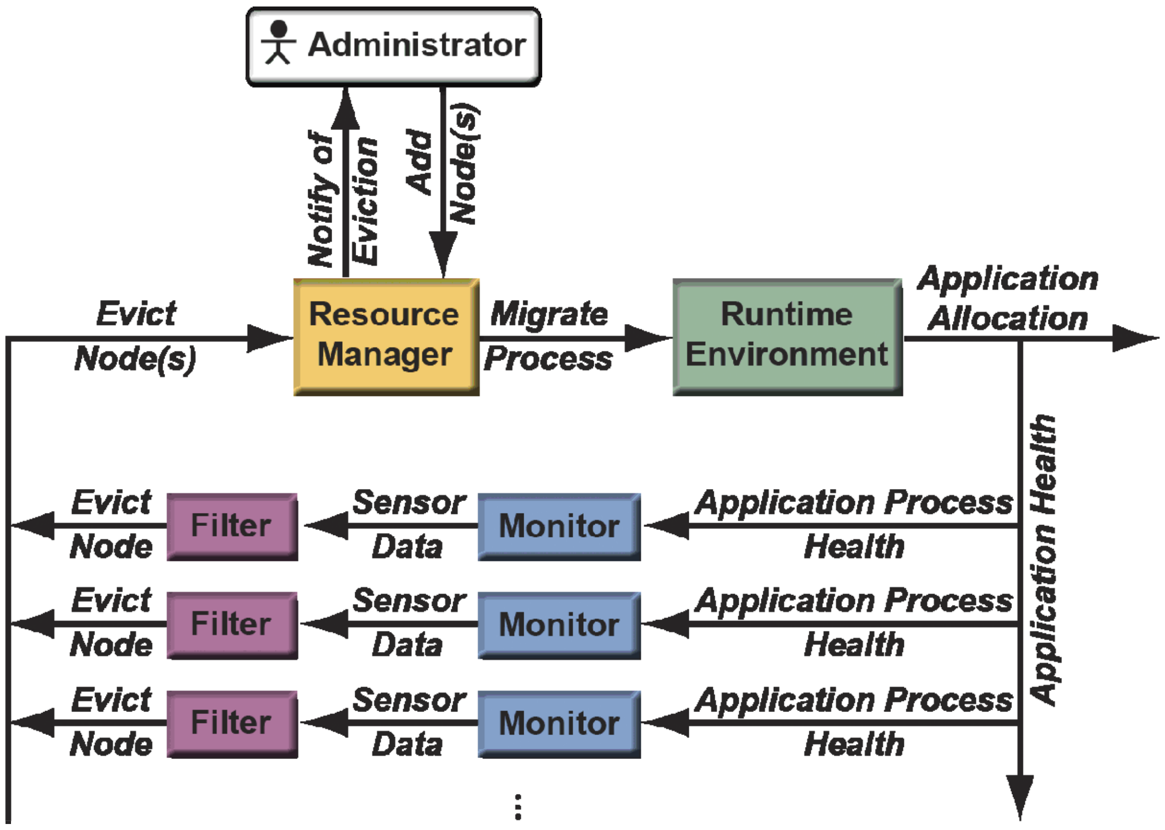


Figure 3.7 Type 2 Feedback-Loop Control Mechanism [22]

The type 2 is slightly different since instead of just notifying the RM upon detecting alerts or exceeding present limits, type 2 utilizes a filter on each node to process raw sensor data and alerts from the MS to notify the RM based on a more thorough analysis of current trends, imminent failures , and possible future threads [22]. After that the RM makes a decision of reallocating the application by migrating from node in use to other spare node. When comparing the 1st and 2nd types, the second one has the better quality of service provided. And moreover, a migration is triggered when a failure is predicted. However is it still may not finish the migrate quickly enough from one node to another one.

Component 4: Reliability and Availability Analysis.

A Reliability Analysis (RA) models the reliability of the system and of each running application. The RA gets the trend data from MS and can make predictions or in the case of actual failure can learn from its mistakes. It is still an open research area. Some future work is discussed in chapter 6. The RA can be as simple as a trend analyser or as difficult as a learning mechanism. The RA requirements for the RAS Framework are: capable of processing environmental system monitoring, system logging, and job and resource monitoring data; store or retrieve data to/from a database. The RA in the RAS Framework is to be used online at the present time and offline using historical data stored in a database.

The type 3 [22] is considered as advanced level of the feedback-loop control mechanism. The mechanism in Figure 3.8 is based on a RA.

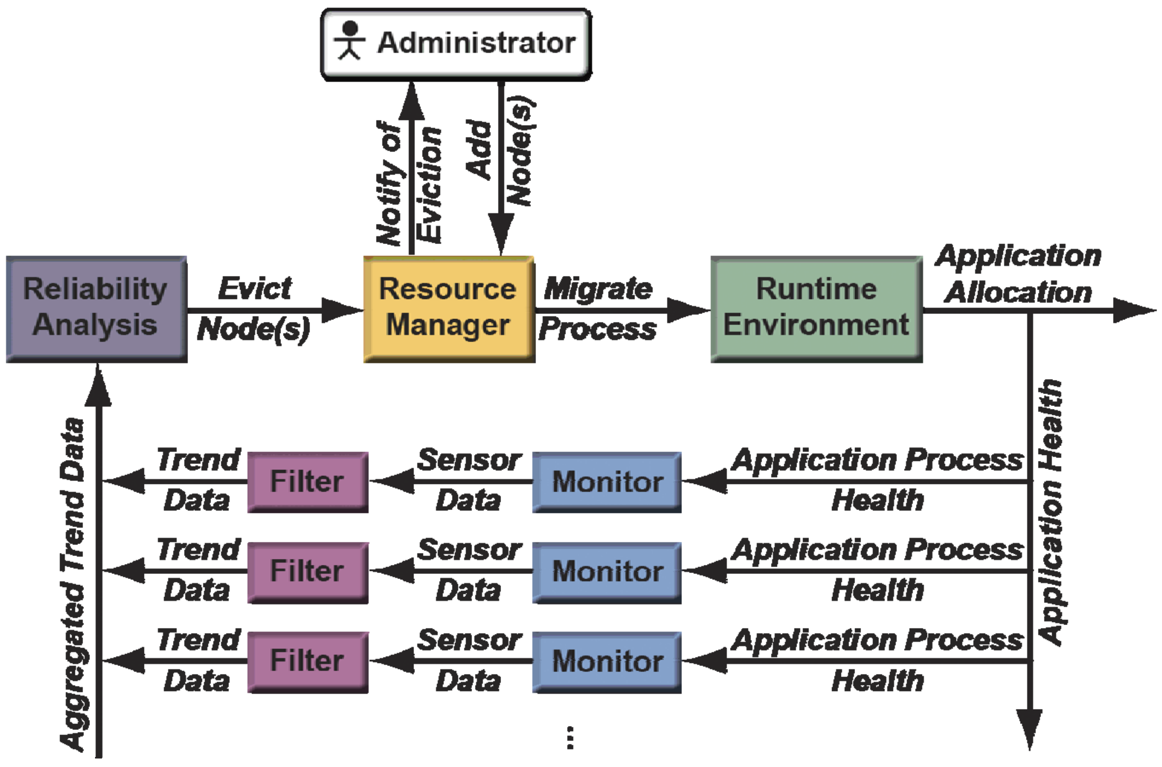


Figure 3.8 Type 3 Feedback-Loop Control Mechanism [22]

The difference between basic and advanced types of the mechanism is a RA, which models the reliability of the system and of each running application. To avoid disruption in the feedback-loop coordination of sensor data, before the filter and trend data after the filter, the RA is needed. The RA alerts the RM to make a decision of reallocating the application by migrating from node in use to a spare node. The type 3 is not able to match system and application reliability patterns because of missing correlation of long-term health context and historical data.

Component 5: Database

A Data aggregation mechanism is required as supporting technology. To maintain knowledge a *database* (DB) can be used. The RA can make some predictions looking at data stored in a DB. A DB can interface directly with used software like MS and then data can be analysed or all the way around as analysed and then stored. Data must be structured in order to be easily queried by the RA. *Database management system* (DBMS) is a mechanism of dealing with data in a DB. There are requirements for the DB and the DBMS to be used in the RAS Framework: fast to query, scalability, widely supported, easy to use, easy to interface with the software, structured data. Examples of the DBMS are: MySQL, Oracle, PostgreSQL. Data can be used online and offline by the RA.

The last type of Proactive FT to be mentioned [22] has even better failure prediction accuracy which improves with long-term system performance from failures. It is as advanced as the type 3 and has an additional DB component. The key point of type 4 is the historical DB. The RA utilizes the historical DB for recording system and applications status and for matching them against the currently experienced pattern in order to optimize the feedback control loop mechanism with learning techniques. Until this project the Types 2, 3 and 4 have not been tested on any systems [22]. The 4th feedback-loop mechanism is discussed further in more details in Chapter 4.

Component 6: Migration Mechanism

When everything is designed in the feedback-loop control mechanism and the failure is predicted it is time to take an action, i.e. to initiate a migration. The Migration in the feedback-loop control mechanism can prolong life of running applications and safe time on preventing the same applications to be computed again and again. Migration requirements for the RAS Framework: fast, making sense - not when it is not needed, like having a spare node, and not after the failure, the RM should support the migration and be aware that a migration is started and finished, but not as a new submitted job, avoid false positive, false negative, true negative, but to have true positive. Migration can be: on a process-level, on a job-level; from a virtual machine to a virtual machine, from a node to a node, from a process to a process [22, 75, 76].

3.4 Components Location

In order to implement the RAS Framework for HPC for production use, it first needs to be tested on small systems, such as clusters. The design of the RAS Framework for HPC systems and clusters might be slightly or completely different. The contribution of this thesis is the RAS Framework Engine Prototype which will be improved over time till the last stage as the RAS Framework. It is difficult to say where each part is located before not developing implementation strategies but it will be discussed in detailed software design, which is located in chapter 5.

3.5 Summary

Since in fault tolerance there is no single solution for dealing with failures the thesis goal is to design a RAS Framework Engine Prototype which consecrates on Proactive fault avoidance by using a migration mechanism. Making a system reliable, available, robust, without failures is a target of the RAS Framework based on Proactive FT and Reactive FT. This chapter provided information about each prototype component.

4. IMPLEMENTATION STRATEGY

“Everything should be made as simple as possible, but no simpler” (A. Einstein)

The implementation strategies of the RAS Framework Engine Prototype are described in this chapter, including use, installation and integration into an operating system, followed with testing strategies. This chapter provides an overview of the information system and includes any additional information that may be appropriate.

Implementation strategies are:

1. Design approach;

2. Development approach;

3. Integration approach;

4. Testing techniques;

5. Testing approach.

4.1 Design Approach

Purpose

The RAS Framework Engine Prototype is a core component of the RAS Framework (Chapter 3). Term Engine is used to describe running a continuous process in order to deal with failures in a proactive way. Before the prototype will be integrated into a HPC system it needs to be tested on clusters. Future work concentrates on the prototype improvements and integration of the extra feature - Reactive FT.

The RAS Framework Engine prototype is based on the 4th type feedback loop control mechanism in order to have more control of a system and to be able to predict basic failures [22].

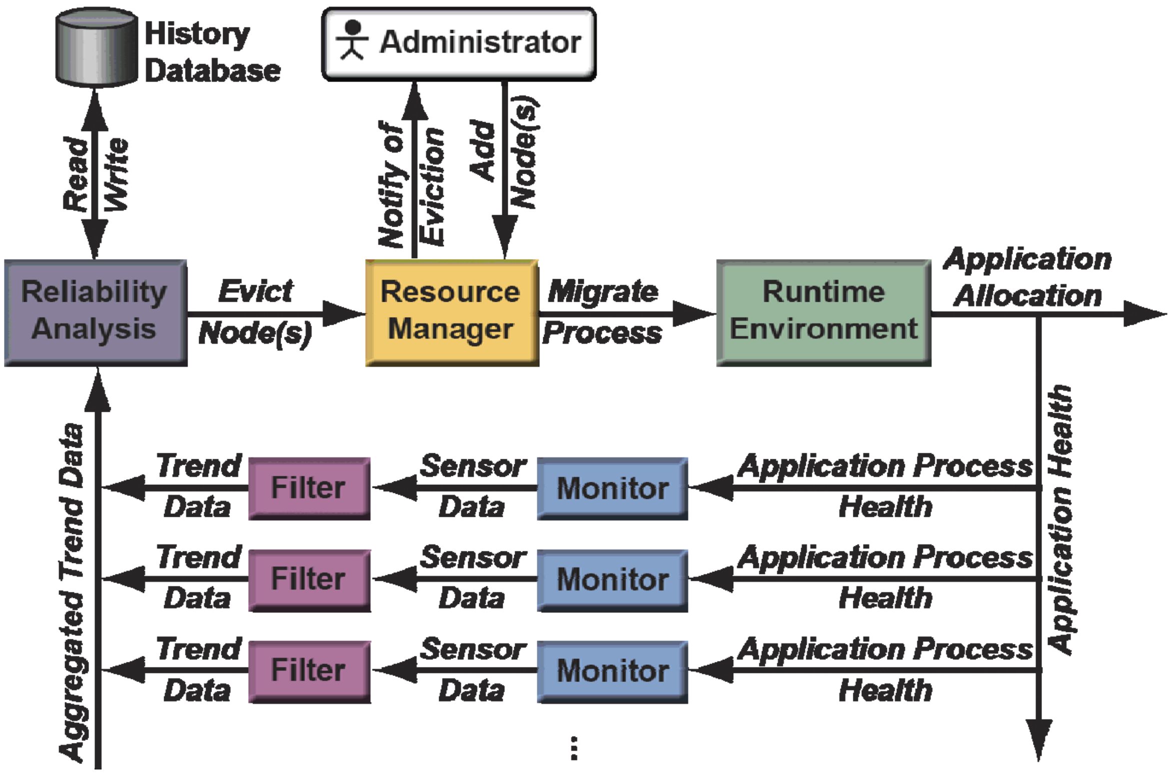


Figure 4.1 Type 4 Feedback-Loop Control Mechanism. [22]

This type 4 feedback-loop control mechanism is an advanced form of Proactive FT and it has even higher quality of service to handle failures than other types 1-3. The mechanism has JS/RM, MS, RA, DB and a migration mechanism [22].

After a job is submitted to the JS/RM, the RM finds computer resources, allocates the job to the resources and deals with the next submitted job. A MS constantly monitors the state of computer resources and aggregates the sensor data and sends the data through filters to a RA to analyse the trend data and/or store it to a DB. The data stored in the DB is periodically analysed by the RA in offline mode. The data which is no longer required or long term data can be archived in order to have available free disk space. Another approach can be taken by storing data to the DB directly from the MS, the RM, System logging and then analysed by the RA [22].

Upon detecting alerts or exceeding thresholds the RA notifies the JS/RM to evict all applications from that particular node or processor and notifies the RE to migrate the applications to a spare node or on a node which is less used. Job parts resume computations upon completion of the migration [22].

4.1.1 Job and Resource Monitoring

Monitoring parallel applications and finding resources for these applications is usually performed by a JS and a RM of a HPC system. The JS software schedules, prioritizes and allocates the given tasks. The RM provides control over batch jobs and compute nodes and finds the resources. Usually the JS and the RM work together as one software but more advanced softwares are splited in two. One has role of schedule jobs and another one of finding computer resources. The JS/RM must satisfy requirements for the RAS Framework Engine Prototype: scalability, widely supported, easy to use, easy to interface with database, support migration. There are different choices to make for the JS and the RM: Portable Batch System (PBS), Torque RM, Maui Cluster Scheduler, Moab Cluster Suite, Slum and others.

PBS [79] is a queuing jobs system, JS, in HPC systems and operated on network multi-platform Unix environments. The main components are the Server *pbs_server*, Scheduler *pbs_sched* and the job executor *pbs_mom*, known as Machine Oriented Mini-Server. And has user, operator and administrator commands. There are several versions:

- Open PBS is a first version of PBS and it is an open source.
- PBS Professional (PBSPro) is a commercial version of PBS, scalable to thousands of nodes and scales hundreds of thousands of nodes.
- Torque RM (is discussed later).

PBS features [80]:

- Wide platforms availability, such as Cray, IBM, Sun;
- Automatic load-leveling;
- File staging;
- Job interdependency:
- Security and authorizations;
- Username mapping;
- Parallel jobs support;
- Job accounting;
- Has a graphical user interface: *xpbs, xpbsmon ;*
- Comprehensive API.

Torque Resource Manager [34] has advanced features in the areas of scalability, fault tolerance, and feature extensions.

Torque is the improved version of OpenPBS in the following areas [34]:

- Fault Tolerance support;
- Scheduling Interface;
- Scalability:
    1. handles larger clusters (over 15 TF/2,500 processors);
    2. handles larger jobs (over 2000 processors);
    3. supports larger server messages;
- Usability.

Torque has the capability of tracking jobs before and after their execution. This capability helps to prepare systems, perform node health checks, prepend and append test to output and error log files, systems cleanup and other [32]. This is done by prologue and epilogue scripts. Some arguments are passed to the prologue and epilogue scripts: job id, user name, user group, job name, list of requested and used resources.

Maui Cluster Scheduler [81] is an open source job scheduler for clusters and supercomputers. Maui is a configurable and optimized tool which is able to support an array of scheduling policies, dynamic priorities, extensive reservations, and has other capabilities. Maui is in use of government, academic, and commercial sites. All of the Maui capabilities are also found in Moab, but Moab has extra features such as virtual private clusters, basic trigger support, graphical administration tools and web-based user portal.

Moab Cluster Suite [35] is an advanced management middleware. The Moab Cluster Suite consists of a Moab Workload Manager – a workload management and scheduling engine; a Moab Cluster Manager - a graphical cluster administration interface, monitor, and management reporting tools; and a Moab Access Portal – a web-base end-user job submission and management portal. Moab can improve performance, increase user productivity, and unite management across cluster.

Moab System Compatibility [35]:

- Operating system support for Linux (all), Unix (AIX, IRIX, HP-UX, FreeBSD, OSF/Tru-64, Solaris, etc.), Mac OSX & limited Windows support
- Resource Manager support for LSF, TORQUE, PBSPro, SGE, SLURM, LoadLeveler, OpenPBS, and custom resource managers

4.1.2 Syslog Logging

Many works rely on system event logs. A user by reading log messages can get information about system usage by other users, softwares, about level of a message, for example warning, error or other information. A failure prediction framework [82] explores log entry correlations. The results showed [82] approximately 76% accuracy in offline predictions and approximately 70% accuracy in online prediction. Other efforts focus on identification of root causes, failure modes, correlations, patterns, trends, failure indication and others [82]. The requirements of Syslog logging for the RAS Framework Engine Prototype: scalability, widely supported, easy to use, easy to interface with a database. There are different choices to make as: system logging daemon (*syslogd*) and kernel logging daemon (*klogd*); system logging ng (*syslog-ng*), *Sysiphus*, and others.

Syslogd [36] and klogd [37] provide support for local and remote login. In Syslogd every logged message contains: time, hostname field, program name field and other. The klogd daemon listens to kernel message sources and is responsible for prioritizing and processing OS messages. The klogd daemon can run as a client of syslogd or as a standalone program.

Syslog-ng [33] is a flexible and scalable audit trail processing tool for different organizations needs. It provides centralised, security stored log of all devices on a network; filtering based on message content; customisable data mining and analysis capabilities.

Features of Syslog-ng [38]:

- Powerful, configurable;
- Filtering based on message content;
- Message integrity;
- Portability ;
- Improved network forwarding.

4.1.3 Environmental System Monitoring

The MS takes responsibility for monitoring many different computer components or software services that are in use and collects metrics values. The MS requirements for the RAS Framework Engine Prototype are: scalability widely supported, easy to use, easy to interface with a database. There are different choices to make: Ganglia, MonAMI, OVIS, Nagios, OpenIPMI, PSP and others.

OpenIPMI [83] has its predecessor an Intelligent Platform Management Interface (IPMI). OpenIMPI goal is to a full-function IPMI system in order to allow full access to all IPMI information on a server and to transport it to a level that will make it easy to use. OpenIPMI consists of two core parts: (1) a device driver, and (2) a user-level library. With OpenIMPI processor temperature and fan speed metrics can be extracted [83].

Ganglia [39,84] is a scalable distributed monitoring system for HPC systems, for example clusters and Grids. Ganglia is based on a hierarchical design of clusters. It uses XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported on different OS and processor architectures, and widely used. It can be scaled to handle clusters with 2000 nodes. Ganglia monitors each node and multicasts the aggregated data to other nodes, thus each node has monitored data from all other nodes. Ganglia exports data in RRD and XML files. RRD files are stored in Round Robin Database (RRD). In *rrd* format only numerical data is collected. This format minimises information for analysis in the RA. The RRD is difficult to query because all the data is stored in a flat file.

MonAMI [25] is a framework which consists of two main parts: MonAMI-core and a set of plugins. The core is a multi-threaded, asynchronous, message-passing framework. The core has many useful plugins as TCP, Torque, Maui, MySQL. It can send gathered data to the MS such as Ganglia, Nagios, MonALISA, KSysguard and R-GMA. A key features of MonAMI is usablity to integrate information coming out from MonAMI into existing monitoring infrastructures.[25]

OVIS [85] is an open source software tool that provides intelligent, real-time monitoring of computer clusters and can also forward collected data to other monitoring system, for statistical processing. There are several OVIS versions, for example 1.1 and 2.

Nagios [40] monitors critical IT infrastructure components, such as system metrics, network protocols, applications, services, servers, and network infrastructure. Nagios provides notifications to an administrator by email, SMS or a script when a critical infrastructure component fail and recovers, after that the administrator can analyse the problem. Reports with outages, events, notifications, and alerts can be reviewed at a later time.

Nagios Features [40]:

- Monitor applications, services, operating systems, network protocols, system metrics and infrastructure components with a single tool;
- Script APIs allow easy monitoring of in-house and custom applications, services, and systems;
- Event handlers allow automatic restart of failed applications, services, and services;
- Scales to monitor more than 100 000 nodes;
- Fail over capabilities, that ensures non-stop monitoring of critical IT infrastructure components.

4.1.4 Database

A Data aggregation mechanism is required as supporting technology. Database Management System (DBMS) is a mechanism of dealing with data in a DB. The DB and the DBMS requirements for the RAS Framework Engine Prototype are: fast to query, scalability, widely supported, easy to use, easy to interface with the softwares, structured data, and hierarchical approach. There are different choices to make: MySQL, Oracle, ProgreSQL and other.

Every source should have a separate table. A DB design is done before the DB creation, including analysing metadata and their types, creating users and their privileges. Main data to be collected from:

- Ganglia - metric names, metric values, node names, node location, time;
- Torque - job name, resources requested, resources provided;
- Syslog-ng - message level, (i.e. information, warning, error), message content.

Before making a choice what DBMS to choose they need to compared. MySQL, Oracle, ProgreSQL are relational DBMS (RDBMS) that efficiently manage data within an enterprise. These RDBMS are available for many hardware platforms and OS; they use Structured Query Language (SQL) to interact with a DB [41]. Oracle supports PL/SQL , XML. MySQL supports C, C++, PHP, Perl, but does not support XML [41]. ProgreSQL is a management system similar to MySQL. A Software [15] can help archiving data in ProgreSQL. The implementation ideas can be used to archive data in a DB using MySQL DBMS.

The software to be used in the RAS Framework Engine Prototype were made towards Torque, Ganglia, Syslog-ng, MySQL. The goal of this thesis is neither to choose the latest version of a software nor to find the best available solution. The goal of a prototype is to show its existence, prove that it is a possible solution for dealing with failures and to show its results. Developers and researchers can improve the prototype performance, make changes in software choices and add extra features. Torque can work together with Moab Workload Manager to improve overall utilization, scheduling and administration on a cluster.

These choices are sufficient enough for the prototype because of the performance, scalability, functionality, robustness, easy to use, open source. It is also worth mentioning that some of these software were used before with other prototypes with Proactive FT. And MySQL DBMS is useful for developing prototype applications or proof of concepts [41].

4.1.5 Reliability and Availability Analysis

A RA can make predictions or in the case of actual failure it can learn from its mistakes. After all the software components are connected the RA takes place of providing a full pre-emptive migration solution. The first set is to check if and how Torque handles process migration, and if not how it needs to be modified in order to be able to do this migration process. Torque, as it was mentioned before, is a system resource manager that keeps track of available resources and their utilization. Process migration changes this mapping and it needs to be considered how Torque handles this task. The RA requirements for the RAS Framework Engine Prototype are: capable of processing environmental system monitoring, system logging and job and resource monitoring data. The RA can be as simple as a deviation trigger or as complex as machine learning mechanism.

Every time a new entry is added to the Ganglia table it is evaluated if it exceeds a threshold that is associated to the metric name. Once a threshold is exceeded, the current list of used nodes is compared with the list of reserved nodes to find a free node, the migration is triggered and changes are recorded in the list of used nodes. It is still an open research area. Some of the future work is discussed in chapter 6. There are at least three ways to consider:

- Everything is done by the scripts that add the Ganglia metrics.
- The script that adds the ganglia metrics calls a user defined *mysql* function to perform the evaluation if migration is needed. If the migration is needed, then the script executes the migration and changes are recorded in the list of used nodes.
- The script that adds to ganglia metrics calls a user defined *mysql* function to perform all the necessary function for evaluation and triggering migration.

4.1.6 Migration Mechanism

Migration approach with the feedback loop control mechanism can prolong the life for running applications and safe time on preventing the same applications to be computed again and again in an event of failure. The Migration requirements for the RAS Framework Engine Prototype are: fast, making sense - not when it is not needed (not whenever there is a spare node), and not after the failure, the RM should support the migration and to know that a migration is started and finished, but not as a new submitted job; avoid false positive, false negative, true negative, but to have true positive. There are different choices to make in order to have a migration process: on a process-level, on a job-level; from a virtual machine to a virtual machine, from a node to a node, from a process to a process. It is still an open research area to make a correct migration for a specific action. Some of the future work is discussed in chapter 6.

4.2 Development Approach

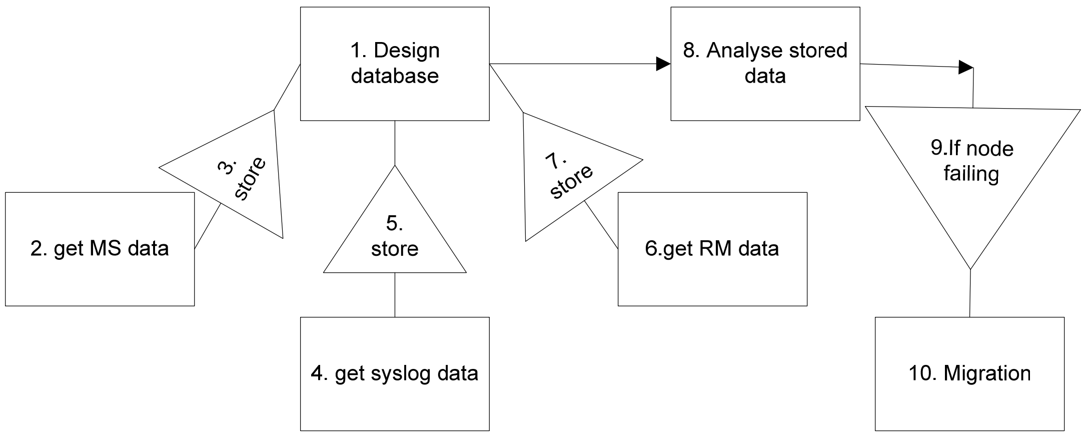


Figure 4.2 Development Steps

Steps to be followed:

1. Create database and users.

2. Create interface between the MS and the DB; create tables; store the data by a daemon.

3. Create interface between System logging and the DB; create a table; store the data by a daemon.

4. Create interface between the RM and the DB; create a table; store the data by prologue and epilogue scripts.

5. Make analysis based on the stored data (still a active area of research); make a prediction.

6. Use the migration mechanism, trigger the migration by a daemon.

Data Structure of the prototype:

- Storing data: the data is stored to a file, log-files and in a DB.
- Organizing data:

  In a file the data is assigned to variables and organized in rows.

  In the DB the data is stored in different types: int, varchar, test and organized in separate tables for Ganglia, Torque and Syslog-ng.

List of languages to be used: the coding will be done using shell script and languages: xsl, sql. The approach is to reuse some code in order to avoid writing the code from the scratch.

List of major functions:

- Shell script functions:

  read/write, copy, test;

  loops;

  make/ remove directory, create/remove a file.

- SQL functions:

  Select; insert; create; update; delete; grant.

Daemons and Their Parts.

Daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events. The daemon does not run under direct control of a user. The *init* process is executed during the booting of the Linux kernel and is responsible for reading instruction from the file /etc/inittab and executing them in order. *Init.sh* a script to start/stop a daemon on the OS boot. *Run.sh* a script to execute a process of a daemon. For creating the daemons and their parts a button-up approach is used.

The RAS Framework Engine Prototype has install, uninstall scripts and README file. Scripts provide functionality to install and uninstall all components step by step and README file guides through the installation process.

Ganglia/MySQL Daemon.

1. In order to have Ganglia/MySQL daemon: write install and uninstall, init and run scripts.

2. In order to have run script as an interface between Ganglia and MySQL: create loop to store data from ganglia to DB periodically.

3. To create a loop: get xml file, use xsl stylesheet and covert to sql with a processor software.

4. Write xsl stylesheet and xsl file contains sql insertion functions.

Syslog-ng/MySQL Daemon.

1. In order to have Syslog-ng/MySQL daemon: write install and uninstall, init and run scripts.

2. In order to have run script as an interface between Syslog-ng and MySQL: create loop to store data from syslog to DB periodically.

3. To create a loop: modify syslog-ng configuration file.

4. To modify syslog-ng configuration file: write additional function to be added to syslog-ng configuration file.

5. Create a pipe for data to go through from syslog to mysql; and an additional function has sql insertion functions.

Torque/MySQL Scripts.

1. In order to have Torque/MySQL scripts: install and uninstall prologue and epilogue scripts are written.

2. Prologue script: check what information Torque gets from a submitted job and provides for prologue script. Prologue has sql insertion function.

3. Epilogue script: check what information Torque gets from a finished or interrupted job and provides for epilogue script. Epilogue has sql update function.

Migration Daemon

1. In order to have a migration daemon: write install and uninstall, init and run scripts.

2. In order to have run script as an interface between DB and migration mechanism: create a loop to analyse data in DB and to trigger a migration if needed.

3. Analyse ganglia and syslog data for migration source, check for spare migration nodes, select migration target, trigger migration, record changes in DB.

4.3 Integration Approach

The first thing to do is to set up a cluster that will run Linux Operating System (OS). The testing resources are: Linux cluster with 48 nodes, Linux desktop machine, Linux cluster with 5 nodes with 4 virtual nodes on each machine. Each node is connected to network File System (NFS).

Steps to follow for the integration:

1. Download, install and configure Ganglia, Torque, Syslog-ng, MySQL.

2. Install a DB and test.

2. Install Ganglia/MySQL daemon and test.

3. Install Syslog-ng/MySQL daemon and test.

4. Install TorqueMySQL scripts and test.

5. Uninstall everything.

6. Install the whole package, uninstall and test.

4.4 Testing Technique

Software testing is one of the main tasks needed to achieve high software quality. Making a plan for testing saves time by avoiding unnecessary tests. Usually a test is on a whole software and on each component. The following figure shows testing fundamentals [26].

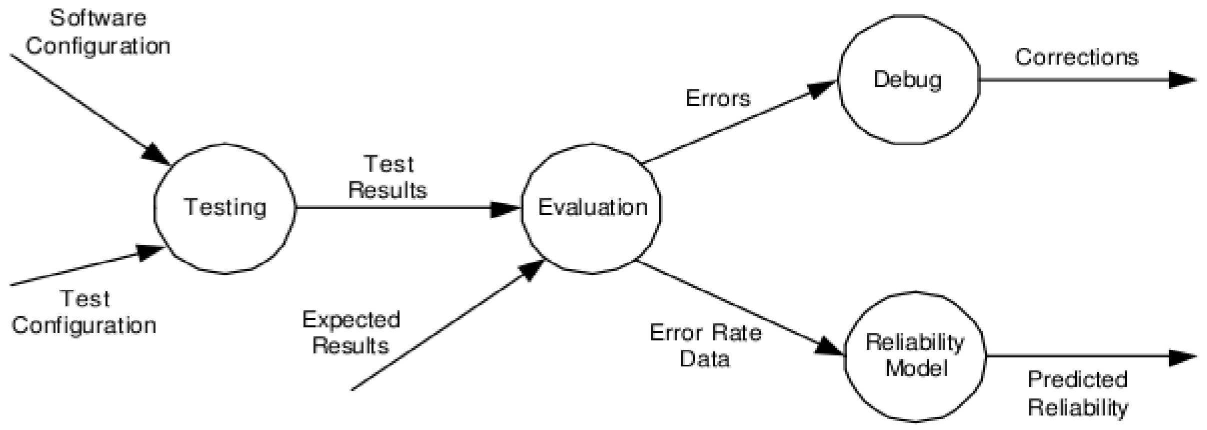


Figure 4.3 Test Information Flow [26].

Two types of testing are proposed: software configuration and test configuration. Results are collected and compared to desired or expected results. In the event of erroneous data being received, further processes take place. The errors can be corrected by hand or by using debugging software [26]. An example of the debugging software is Distributed Debugging Tool (DDT) [16]. Another solution is to use reliability model.

4.4.1 White Box Testing

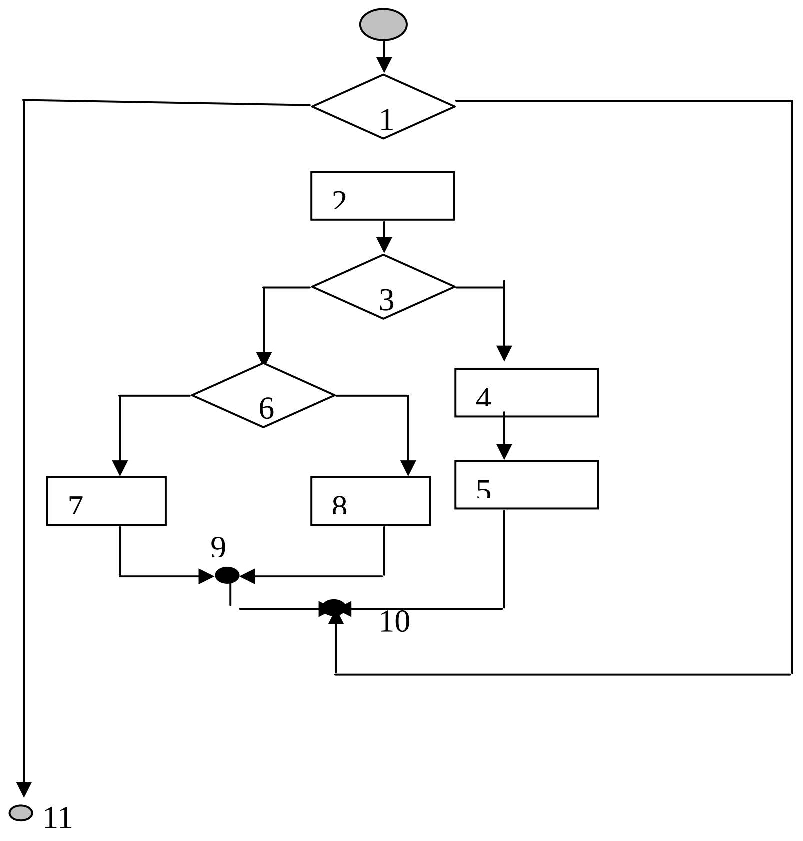


Figure 4.4 Flow Chart [26]

On a general view many different steps can be takes as: 1,2,3,6,7,9,10,1,11 or 1,2,3,4,5,10,1,11 and so on in order to test the whole software. This technique can be used to guarantee that all independent parts are exercised at least once, testing logical decisions, execute loops and exercise data structure for their validity [26].

4.4.2 Black Box Testing

This technique provides additional testing to the white box testing and different types of errors are examined. Categories are: incorrect or missing functions, interface errors, external DB access, performance errors, initialization and termination errors [26].

4.5 Testing Approach

A software test strategy guides the software developer, the quality assurance organization and the customer on a road map. Any testing approach needs to be well planned, designed, executed and evaluated. A strategic approach to software testing starts with software design and works towards the integration of the complete system. Some of the possible and appropriate tests are: unit testing, integration testing, validation testing, system testing and debugging [26].

4.6 Summary

This chapter provides the requirement and specification strategies for the RAS Framework Engine prototype. Difference approaches are considered, discussed and analysed. This chapter guides from design and development approaches and concludes with integration and testing techniques for the prototype.

5. DETAILED SOFTWARE DESIGN

Details of the planning and requirement analysis are discussed in the previous chapter for the RAS Framework Engine Prototype. This chapter consists of the prototype development life cycle. It includes initiation, the prototype limitations, detailed design, development steps and integration steps, as well as testing. Test results are shown in the Appendices, and the conclusions and future work are discussed in chapter 6.

5.1 Initiation

The presented RAS Framework Engine Prototype was implemented and tested on the Linux operating system (OS), as most HPC systems are running the Linux OS on all nodes or at least on service nodes. MySQL database management system was chosen for the database. Ganglia, Torque, Syslog-ng softwares were chosen for environmental monitoring, job and resources monitoring, and event logging respectively. In addition, numerous stateless daemons, i.e. processes and scripts, were developed to provide interfaces between the database and the softwares. These daemons are able to store aggregated data to the database via Structured Query Language (SQL) for later analyses by the Reliability Analysis mechanism. The aim of the RAS Framework Engine Prototype is to improve fault tolerance in HPC systems.

5.2 Prototype Limitations

The RAS Framework Engine prototype has limitations in the Reliability Analysis (RA) as it is a future work project. Reactive FT approach is not used but can be easily added with checkpoint/restart capability. The prototype was tested on a cluster, but not on a production-type HPC system as the software is not mature enough. There is currently no support for Microsoft Windows systems.

Speaking about time as stored data and time used for making predictions, the time has drifting issues because it is very difficult to coordinate the time between created intervals. For example, whenever the predefined interval has finished and data is stored, one daemon, i.e. process starts checking information located in the database in one table and updates data located in another table. The process can take a couple of seconds. After the daemon finishes this process the next interval starts. Difficulties are found in missing seconds. The time is stored in unix time, which is in seconds (ten digits number).

5.3 Design

The designed RAS Framework Engine Prototype is implemented on Linux OS. The prototype has individual components: Torque - a Resource Manager (RM) for scheduling jobs and finding resources of the jobs; Ganglia as Monitoring System (MS) for monitoring resources (Ganglia uses the Ganglia Monitoring daemon (gmond) only); Reliability Analysis to analyse data and make predictions; MySQL database management system (DBMS); database (DB) is called *ras* DB as a data storage; Event Logging mechanism is syslog-ng; and has daemons: gangliamysqld, torquemysql, syslogmysqld and migrationd. These stateless daemons are developed to provide interfaces between the DB and the software. The location of each component is shown below on a head node and on each compute node.

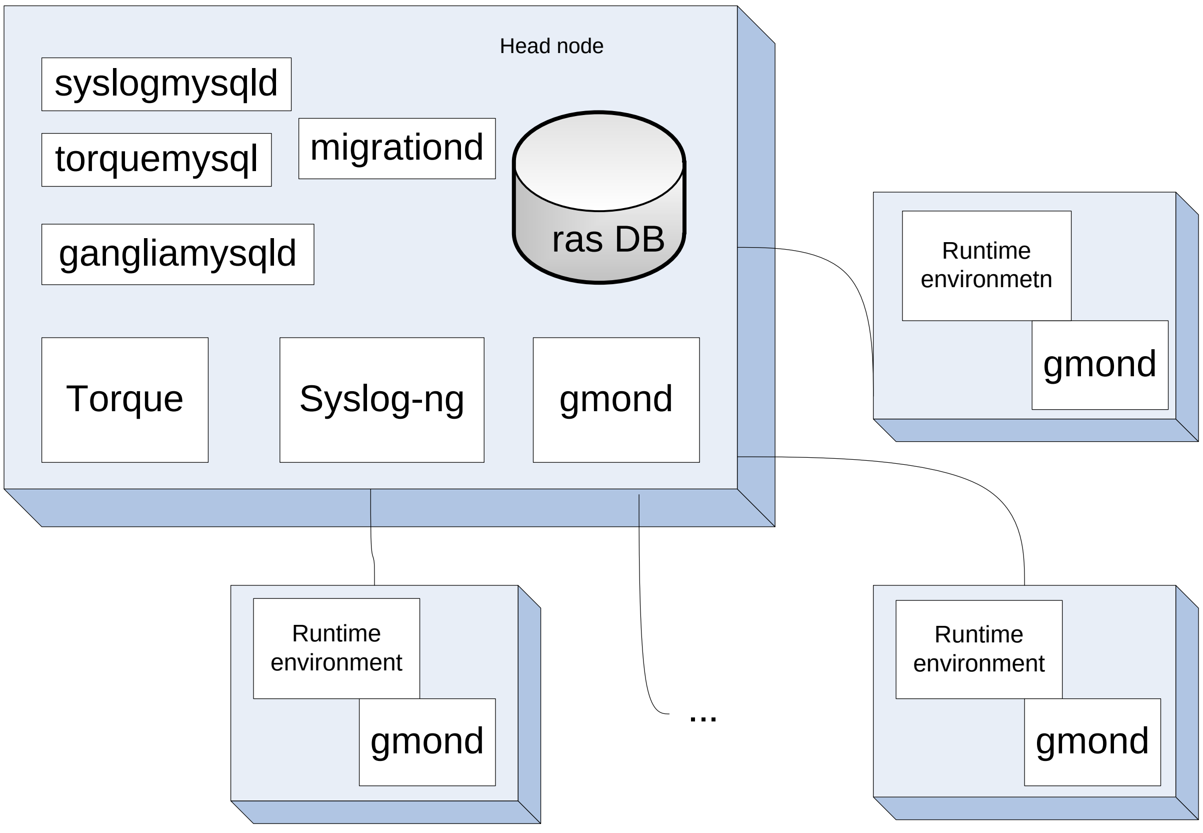


Figure 5.1 RAS Framework Engine Prototype

The design of the prototype is slightly different from Figure 4.1 (previous chapter), meaning that the data is stored from Ganglia, Torque and Syslog-ng and not from RA at the first instance. This decision was made in order to build a foundation for creation of the RA. In this case future work on analysing data can be started in offline mode.

Data Structure.

In the prototype software the data is stored to a file, log-files and in *ras* DB. The data is organized in the following way:

- In a file the data is assigned to variables and organized in rows.
- In *ras* DB the data is stored in different types: int, varchar, test;

  and organized in tables: *ganglia, nodesstate, nodeshistory; torque; syslog.*

5.3.1 Database

A database is a collection of information stored on computer storage device. The data is organized in a way that a DBMS, in this case MySQL DBMS can quickly select, insert, update or delete desired pieces of data. The data is grouped into tables by the used softwares. Figure 5.2 shows the overall structure of thesis DB.

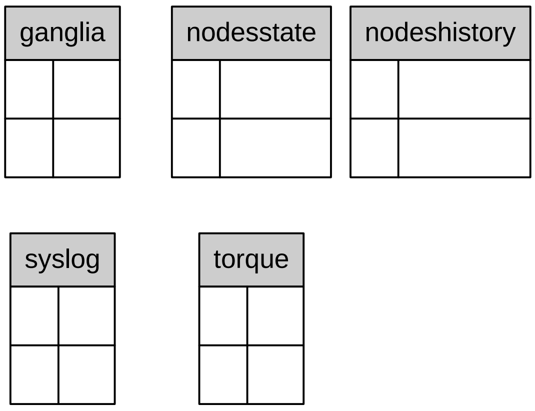


Figure 5.2 *ras* Database

The data is stored in *ras* DB from Ganglia, Torque and Syslog sources. The DB consists of 5 tables and is located on the head node. Each source, Ganglia, Torque and Syslog has different parameters and different data types. Each of these sources has a separate table, however ganglia has three tables. This decision was made after looking on actual software and what data it has to be collected. There are no relationships between tables in *ras* database, i.e. a primary key in one table is not a secondary key in another table. Making relationships between tables will decrease performance, because the collected data must be quickly inserted and stored data must be quickly retrieved.

Ganglia tables:

- *Ganglia* table has a sequence number, metrics names, values of metrics, metric type, value of metric unit, TN – number of seconds since the metric was last updated, TMAX – freshness of a metric, DMAX – number o seconds gmond retains an old metric, slope (i.e. zero or both), source (i.e. gmond or gmetric - another tool to collect metrics), node names, and the time when a metric was collected and the time when a metrics was inserted to the DB.

- *Nodesstate* table consists of node names, time when an xml file was collected from gmond for the first time and time when an xml file was collected from gmond for the last time. The last value in the table is updated whenever ganglia sends new data.

- *Nodeshistory* table has a sequence number, a node name, data about nodes status: either a node is up (value 1 will be inserted) or down (value 0 will be inserted) over a period of time. These analysis is done by looking at *nodesstate* table.

Torque table:

- *Torque* table has information about job submission: job index, job id, user id, group id, name of a job, session id, resources requested and resources allocated, queue type (i.e. serial or parallel), an account if provided, and the time when a job started and finished or aborted.

Syslog-ng table:

- *Syslog* table provides information about a sequence number, node name, facility, priority, level of information (i.e. warning, error), time when it was collected and the time when it was stored, programme name and message information.

5.3.2 Environmental Monitoring

The environmental monitoring component consists of Ganglia MS; to be more precise only the Ganglia monitoring daemon (*gmond*) is used, and a stateless daemon, *gangliamysqld*, that queries data from Ganglia and stores it to *ras* DB at predefined intervals. The data, such as values of cpu speed or cpu temperature, is collected locally by gmond on each compute node and is broadcasted to other nodes at regular intervals. The standard configuration of Ganglia allows gangliamysqld to query gmond on port 8649 at regular intervals in order to collect aggregated values of metrics and to store them to the DB. Ganglia provides monitoring data in Extensive Markup Language (XML) format and gangliamysqld converts this format into SQL statements to store the data. The gangliamysqld must reside on the same node as gmond. However it is not necessary to have gangliamysqld and the ras DB on the same node, because gangliamysqld can connect to the node remotely. Gmond is shown in Figure 5.3

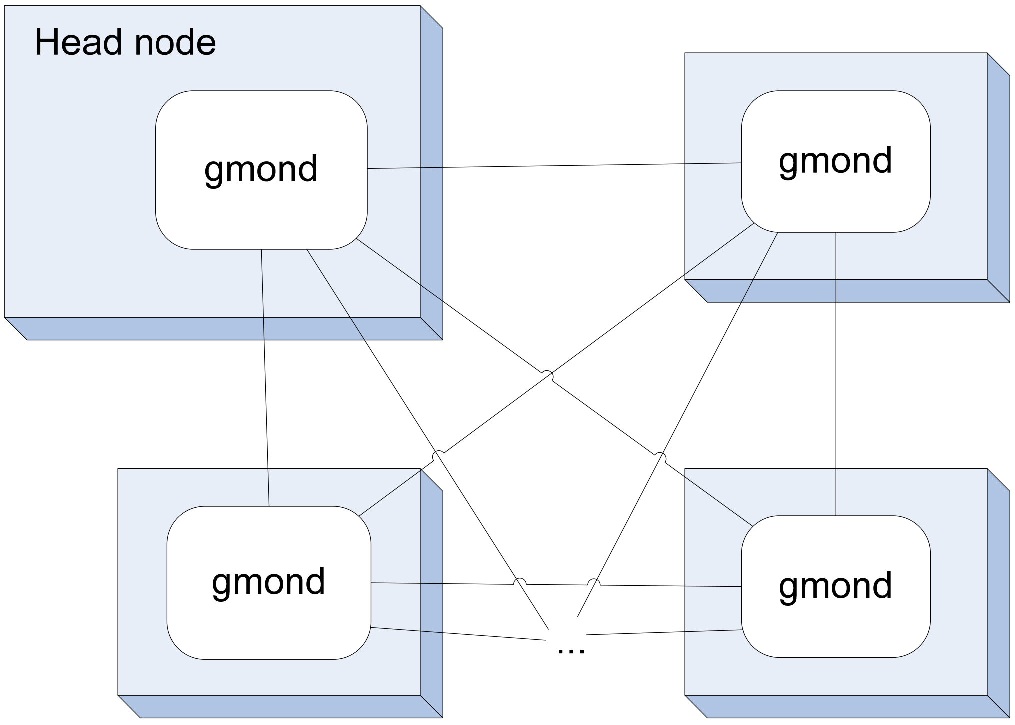


Figure 5.3 Ganglia Monitoring Daemon on Each Node

Every gmond is located on each node and collects the values of metrics and then broadcasts to other gmond. In this way every node has the information of the metrics of values about other node metrics. This approach is not scalable, but some future work is proposed in chapter 6.

All gathered data is stored with time stamps (unix time): (1) when the data was collected and (2) when the data was stored. The difference between stored and collected time can be a couple of seconds. The stored data is stored in raw format. i.e. no pre-processing is done. At the moment it is not known what type of pre-processing of the data it makes sense to use in Proactive FT approach. This is a future research work.

5.3.3 Job and Resource Monitoring

The job and resource monitoring component of the RAS Framework Engine Prototype consists of Torque RM and two stateless scripts: *prologue* and *epilogue*. Torque is based on PBS. To submit a single or multiple processor job to PBS a script must be written. This script includes such information as a name of the program, memory, wall time, processor requirements of the job, which queue it should run in, i.e. s*ingle* or *parallel,* and how a user wants to be notified about the results of the job, i.e. by email. The script is executed by a *qsub* command. *Qstat* command provides information about submitted jobs.

For example, a script consists of:

- Declaration of a job name,
- Name of queue to be run in,
- Number of nodes and number of processors requested,
- CPU time requested,
- Notification method on job completion or abortion,
- Email address,
- Directory from where the job will be submitted,
- Name of the program to be run with parameters.

Torque RM has the capability of tracking jobs by two scripts: prologue and epilogue. A prologue script [32] is executed after a job is submitted to the RM and before the job is allocated on resources. The prologue script stores into ras DB all information about jobs that were submitted, and information about resources that will be in use. After a job is submitted and finished or aborted, Torque executes another script, epilogue [32]. (Upon failed job the RM is notified and epilogue scripts gets information about failed job). The epilogue script stores into ras DB all information about finished or aborted jobs and information about resources that were in use. The prologue and epilogue scripts must run on the same node as Torque, *pbs-mom* server. However it is not required to run prologue and epilogue scripts on the same node with ras DB, because a connection can be done remotely.

5.3.4 Message Logging

Even logging, also called message logging, component of the developed RAS Framework Engine Prototype consists of the Syslog-ng event logging system and a stateless daemon, syslogmysqld, that forward data from syslog-ng system to ras DB at regular intervals. Syslog-ng is an advanced form of message logging and provides events information about an OS, daemons warnings, error messages and other. Event log data is gathered locally on a node by syslog-ng daemon and forwarded to a central syslog-ng daemon on service or head node at predefined intervals [33]. All these messages are written to a pipe, /var/syslogmysqld [33], and are read out by syslogmysqld for an action to store to ras DB. The data gathered from syslog-ng is inserted in SQL statements in order to transfer the data to ras DB. The syslogmysqld deamon needs to be on the same node as syslog-ng. However they can be on a different node than ras DB is on.

5.3.5 Reliability Analysis

The RA is still an open research area but one of the solutions is to use a ganglia scripts for storing ganglia metrics and upon reaching the thresholds execute the migration mechanism. Because many failures are related to a specific system and software failures related to a software that was used on that specific system, the RA must be designed according to the system. In this project for the RAS Framework Engine Prototype the RA was not developed, because of the limitation of the project. Combined environmental monitoring and event logs the RA for failure prediction is not implemented yet due to the missing historical data and tests. At the moment the monitored data is stored but not yet analysed.

The migrationd and ras DB can be on different nodes and migrationd can connect to ras DB remotely. However the migrationd needs to be on the node from where the migration mechanism is executed.

5.3.6 Migration Mechanism

The migration component is based on the Berkeley Laboratory checkpoint/restart (BLCR) layer for LAM/MPI, (previous work [14]). The migration is done at process-level. Whenever the RA predicts a failure, it triggers a migration from one compute node to a spare node. The migration is invoked from the command line by the stateless daemon, migrationd.

5.4 Development

List of languages: the prototype were written using shell scripts, languages: xsl, sql

List of major functions:

- Shell script functions:

  echo; read; cpush; test; ps; grep; cut; cat; nc; update-rc.d; sleep; cexec; ssh; kill;

  if... else; case; while... do;

  scp; cp; mkdir; rm; mv; mkfifo;

  mkdir, rm, mv.

- SQL functions:

  select; insert; create; update; delete; grant;

  unix_timestamp(), from_unixtime().

- XSL functions:

  <xsl: stylesheet>;

  <xsl: template>;

  <xsl: call-template>;

  <xsl: for-each>;

  <xsl: text>;

  <xsl: value-of>;

  <xsl: if test>;

  position(); last().

The steps of ras DB creation:

- create *ras* DB;
- create 2 users: *rasrw, rasro* and grand privileges;
- create tables with install.sh script.

5.4.1 Interfaces

List of Interfaces between:

- Ganglia - DB;
- Syslog-ng – DB;
- Torque – DB.

Ganglia – DB interface:

- Create xsl file;
- Using XSLTproc: xsltproc .xsl .xml > .sql;
- Create loop to store xml files at the regular intervals.

Syslog-DB interface:

- Collect all log files from syslog-ng;
- Send it to a pipe file;
- Transfer the data from the pipe to DB at predefined intervals;
- Wait for the next interval.

Torque – DB interface: whenever Torque is ready to execute a job it runs proloque.sh to store data before a job is allocated on compute nodes. And whenever a job is done or aborted, Torque gets notification about this from the system and executed epiloque.sh. This script updates *torque* table in *ras* DB with used resources and finished time information.

5.4.2 Daemons

List of daemons:

- gangliamysqld;
- syslogmysqld;
- torquemysql scripts;
- migrationd.

Ganglia/MySQL daemon is shown in Figure 5.4.

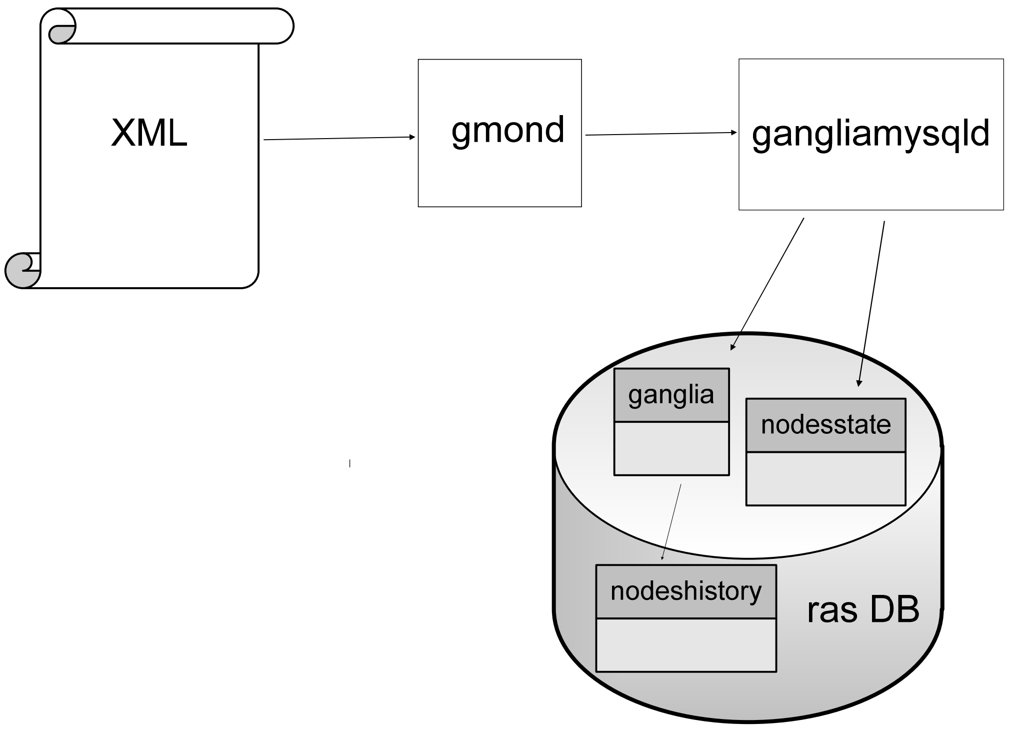


Figure 5.4 Ganglia/MySQL Daemon

Scripts of Ganglia/MySQL daemon:

- gangliamysqld.xsl a style sheet for xml file;
- gangliamysqld-init.sh a script that tells the OS to execute ganglia daemon upon a system boot;
- gangliamysqld-run.sh an interface between ganglia and ras DB;
- install.sh to install Ganglia/MySQL daemon;
- uninstall.sh to uninstall ganglia/MySQL daemon.

Syslog/MySQL daemon is shown in Figure 5.5.

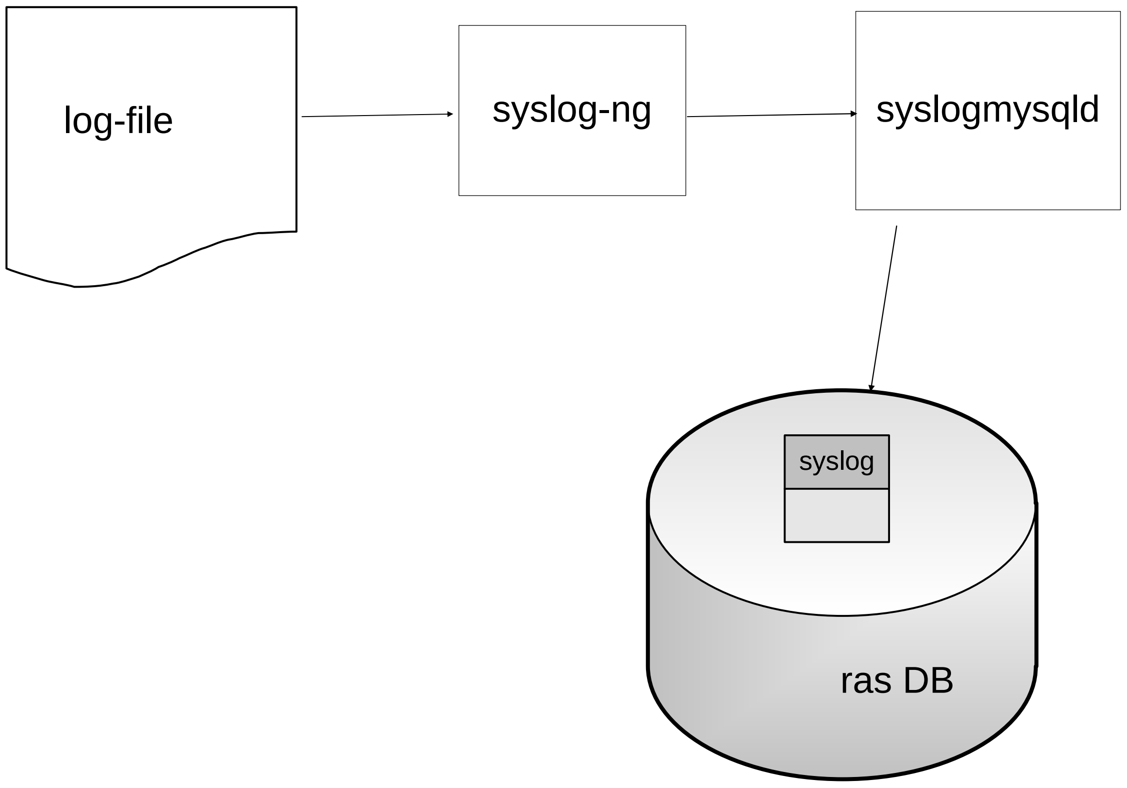


Figure 5.5 Syslog-ng/MySQL Daemon

Scripts of Syslon-ng/MySQL daemon:

- syslogmysqld.conf to modify syslog-ng configuration file;
- syslogmysqld-init.sh script that tells the OS to execute syslog/MySQL daemon upon a system boot;
- syslogmysqld-run.sh an interface between ganglia and ras DB;
- install.sh to install Syslog/MySQL daemon;
- uninstall.sh to uninstall Syslog/MySQL daemon.

Torque/MySQL scripts are shown in Figure 5.6.

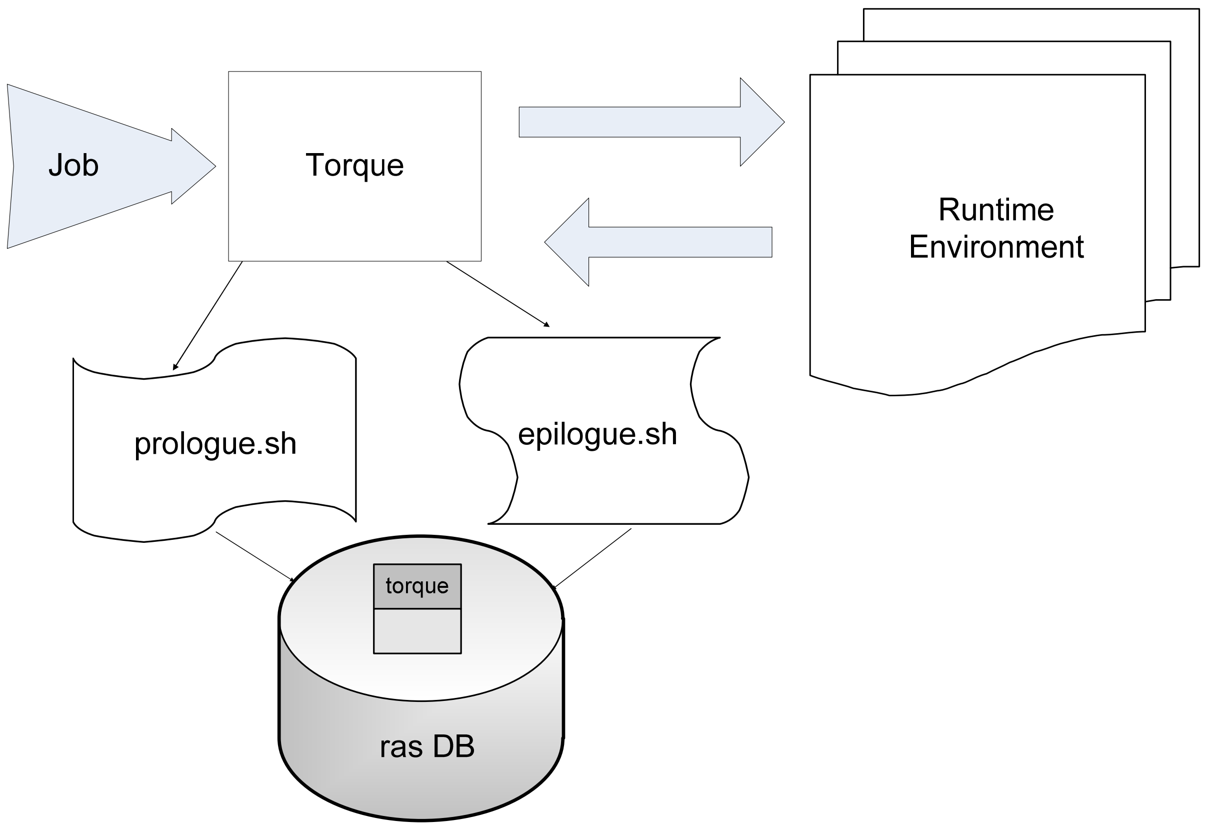


Figure 5.6 Torque Scripts

Scripts of Torque/MySQL scripts:

- prologue.sh to store information in DB about a job before execution;
- epilogue.sh to update information in DB about the job after completion;
- install.sh to install Torque/MySQL scripts;
- uninstall.sh to uninstall Torque/MySQL scripts.

Migration daemon is shown in Figure 5.7.

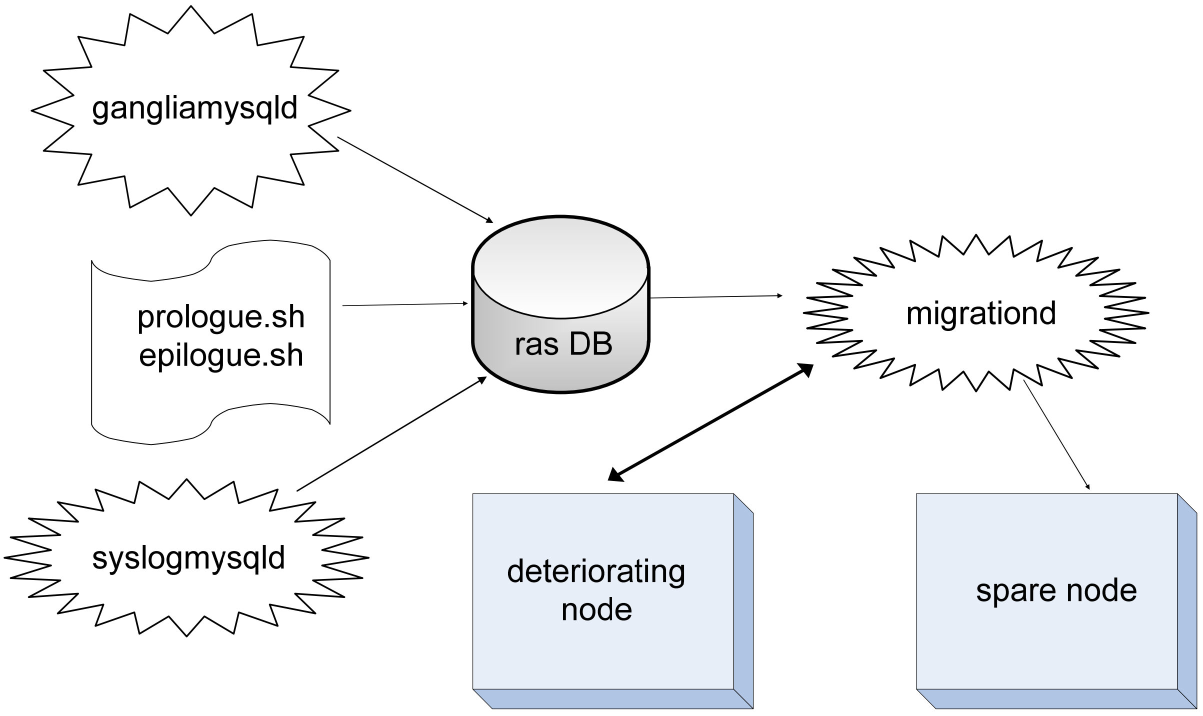


Figure 5.7 RAS Framework Engine Prototype with Migration Daemon

Scripts of Migration daemon:

- migrationd-init.sh a script that tells the OS to execute migration daemon upon a system boot;
- migrationd-run.sh an interface between the RA and migration process;
- install.sh to install migration daemon;
- uninstall.sh to uninstall migration daemon.

5.5 Integration

The RAS Framework Engine Prototype was integrated on a Linux cluster, with Ubuntu 8.10 and 9.04 Linux versions (other versions were not tested). Before installing the RAS Framework Engine Prototype other softwares: MySQL on the head node, gmond from Ganglia (on each compute node), Torque (pbs_server on the head node, pbs_mom on each compute node), Syslog-ng (on each compute node) are installed and configured on the cluster. The prototype consist of 3 files: to install the software, to uninstall and README document (can be found on the attached CD).

The prototype can be installed by a user with high privileges (root) in order not to corrupt the cluster. Installation process consists of installation steps of components of the prototype. During the installation a user is notified about installation details and about the completion. The uninstallation process is a reverse process: uninstalls the last installed component followed to the first installed and notifies the user upon completion. README file serves as user's manual and guides a user through the installation and uninstallation process (can be found on the attached CD and in Appendices: User's Manual). All the prototype components can be installed separately but a strict order should be followed.

5.6 Testing

Testing was done on a 1 node computer and on a 48 node cluster 'XTORC' located at ORNL. This is a scientists cluster for experimental purposes. The test suite is provided in Appendices: Test Suite.

5.7 Summary

This chapter describes the RAS Framework Engine Prototype design with all its components, as well as development, integration and testing of the prototype. Before having the prototype in a production use more improvements and tests should be made; more details are in the chapter 6.

6. CONCLUSION

“Perfection is reached not when there is no longer anything to add, but when there is no longer anything to take away”. (A. Saint-Exupery)

In this chapter the results, evaluation, critical comments and future work of the RAS Framework Engine prototype can be found.

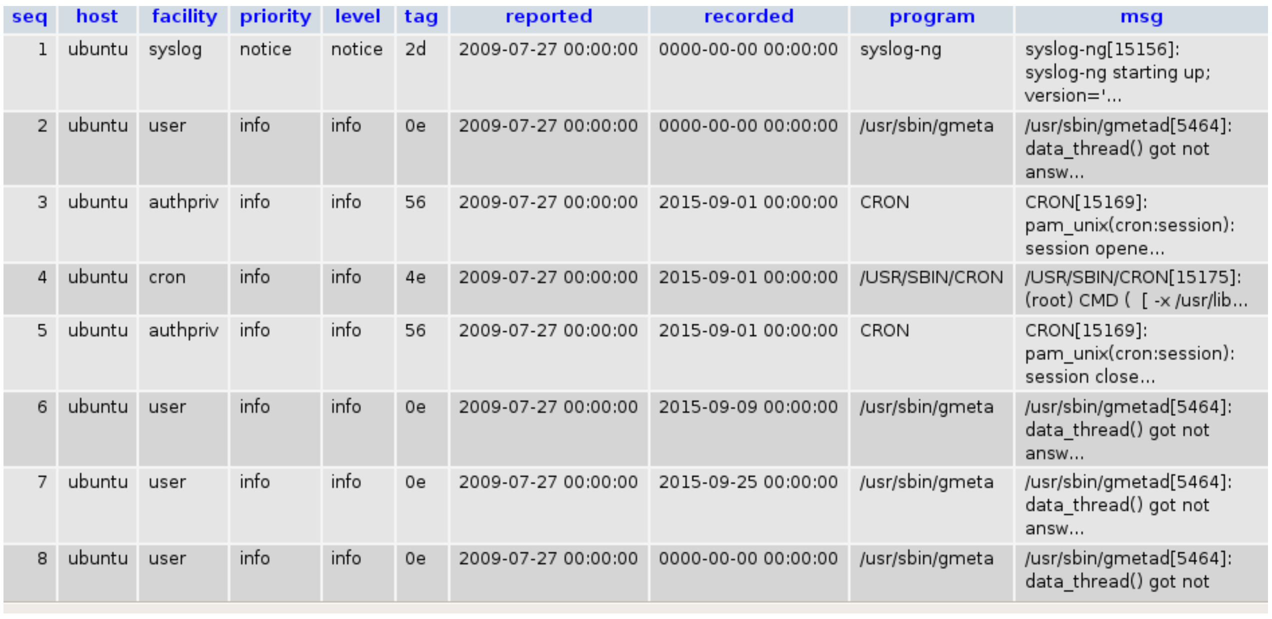
6.1 Evaluation

The thesis presents the first developed RAS Framework based on the type 4 feedback-loop control mechanism for HPC systems. The RAS Framework Engine Prototype was developed and deployed on the XTORC Linux cluster at Oak Ridge National Laboratory. The cluster consists of a head node and 48 compute nodes. The job and resource manager Torque, Ganglia Monitoring System and Syslog-ng event logging mechanism run on all nodes. Stateless daemons: of the environmental monitoring component, gangliamysqld and reliability analysis component, migrationd were deployed on the head node; the event logging component, syslogmysqld were deployed on a separate service node. Stateless scripts of the job and resource monitoring components were deployed on the head and compute nodes. The RAS database was located on the head node.

The developed RAS Framework Engine Prototype is able to collect environmental system monitoring, system event logging, parallel job monitoring and system resource monitoring data from Torque, Ganglia and Syslog-ng and to store the data to ras DB through SQL statements. Further work needs to be done on analysing the aggregated data to predict failures. At the moment it is impossible to make predictions because care and long-term observations need to be done by statisticians. The reliability analysis component creation is not included in the boundaries of this thesis.

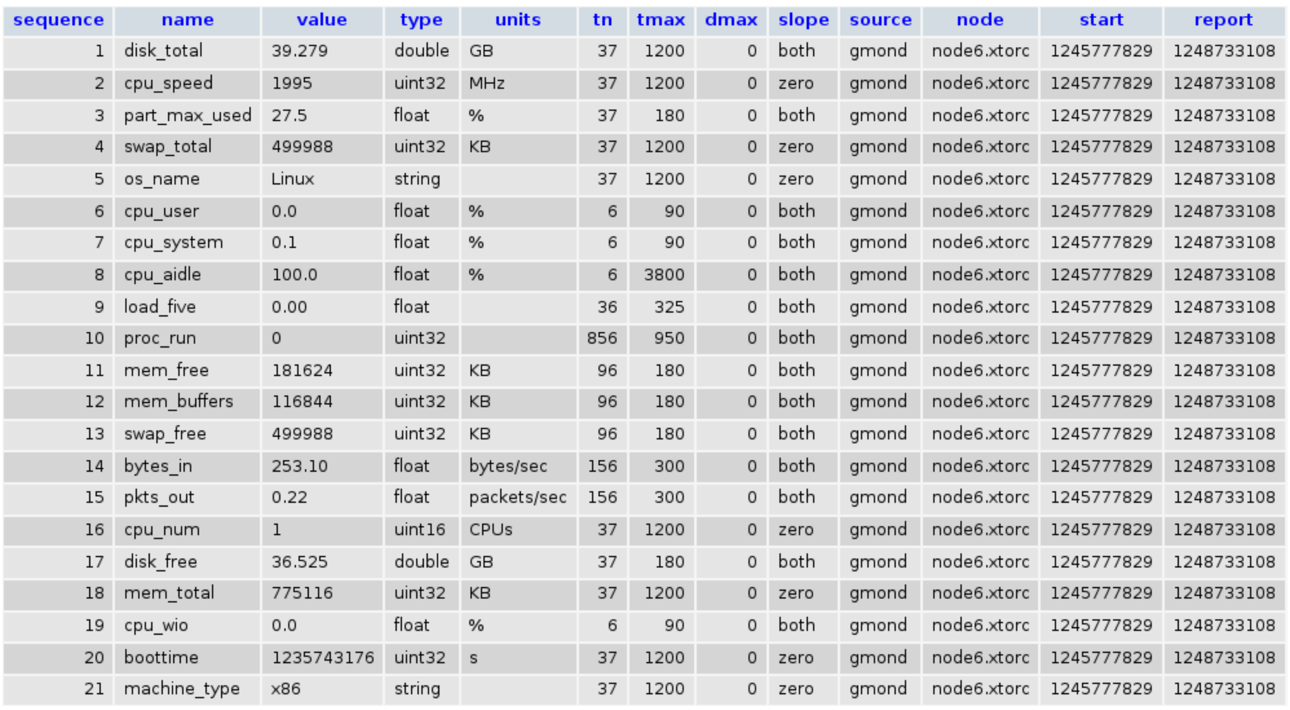
6.2 Results

Because the DB does not have relationships between tables this increase the performance in querying, inserting, deleting and updating the tables. Some of the stored data in the *ras* DB can be seen below.



seq	host	facility	priority	level	tag	reported	recorded	program	msg
1	ubuntu	syslog	notice	notice	2d	2009-07-27 00:00:00	0000-00-00 00:00:00	syslog-ng	syslog-ng[15156]: syslog-ng starting up; version='...
2	ubuntu	user	info	info	0e	2009-07-27 00:00:00	0000-00-00 00:00:00	/usr/sbin/gmeta	/usr/sbin/gmetad[5464]: data_thread() got not answ...
3	ubuntu	authpriv	info	info	56	2009-07-27 00:00:00	2015-09-01 00:00:00	CRON	CRON[15169]: pam_unix(cron:session): session opene...
4	ubuntu	cron	info	info	4e	2009-07-27 00:00:00	2015-09-01 00:00:00	/USR/SBIN/CRON	/USR/SBIN/CRON[15175]: (root) CMD ([-x /usr/lib...
5	ubuntu	authpriv	info	info	56	2009-07-27 00:00:00	2015-09-01 00:00:00	CRON	CRON[15169]: pam_unix(cron:session): session close...
6	ubuntu	user	info	info	0e	2009-07-27 00:00:00	2015-09-09 00:00:00	/usr/sbin/gmeta	/usr/sbin/gmetad[5464]: data_thread() got not answ...
7	ubuntu	user	info	info	0e	2009-07-27 00:00:00	2015-09-25 00:00:00	/usr/sbin/gmeta	/usr/sbin/gmetad[5464]: data_thread() got not answ...
8	ubuntu	user	info	info	0e	2009-07-27 00:00:00	0000-00-00 00:00:00	/usr/sbin/gmeta	/usr/sbin/gmetad[5464]: data_thread() got not

Table 6.1 Syslog-ng Data in the Table *syslog*



sequence	name	value	type	units	tn	tmax	dmax	slope	source	node	start	report
1	disk_total	39.279	double	GB	37	1200	0	both	gmond	node6.xtorc	1245777829	1248733108
2	cpu_speed	1995	uint32	MHz	37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
3	part_max_used	27.5	float	%	37	180	0	both	gmond	node6.xtorc	1245777829	1248733108
4	swap_total	499988	uint32	KB	37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
5	os_name	Linux	string		37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
6	cpu_user	0.0	float	%	6	90	0	both	gmond	node6.xtorc	1245777829	1248733108
7	cpu_system	0.1	float	%	6	90	0	both	gmond	node6.xtorc	1245777829	1248733108
8	cpu_aidle	100.0	float	%	6	3800	0	both	gmond	node6.xtorc	1245777829	1248733108
9	load_five	0.00	float		36	325	0	both	gmond	node6.xtorc	1245777829	1248733108
10	proc_run	0	uint32		856	950	0	both	gmond	node6.xtorc	1245777829	1248733108
11	mem_free	181624	uint32	KB	96	180	0	both	gmond	node6.xtorc	1245777829	1248733108
12	mem_buffers	116844	uint32	KB	96	180	0	both	gmond	node6.xtorc	1245777829	1248733108
13	swap_free	499988	uint32	KB	96	180	0	both	gmond	node6.xtorc	1245777829	1248733108
14	bytes_in	253.10	float	bytes/sec	156	300	0	both	gmond	node6.xtorc	1245777829	1248733108
15	pkts_out	0.22	float	packets/sec	156	300	0	both	gmond	node6.xtorc	1245777829	1248733108
16	cpu_num	1	uint16	CPUs	37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
17	disk_free	36.525	double	GB	37	180	0	both	gmond	node6.xtorc	1245777829	1248733108
18	mem_total	775116	uint32	KB	37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
19	cpu_wio	0.0	float	%	6	90	0	both	gmond	node6.xtorc	1245777829	1248733108
20	boottime	1235743176	uint32	s	37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108
21	machine_type	x86	string		37	1200	0	zero	gmond	node6.xtorc	1245777829	1248733108

Table 6.2 Ganglia Data in the Table *ganglia*

nodename	start	updated
node1	1249600000	1249600819
node2	1249600000	1249601047
node3	1249600000	1249600819
node4	1249600000	1249601047
node5	1249600000	1249600819
node6	1249600000	1249600819
node7	1249600000	1249601047
node8	1249600000	1249600819
node9	1249600000	1249601047

Table 6.3 Ganglia Data in the Table *nodesstate*

sequence	nodename	status	timestamp
1	node1	1	1249600811
2	node5	1	1249600811
3	node7	1	1249600811
4	node10	1	1249600811
5	node4	0	1249600811
6	node4	1	1249600811
7	node6	1	1249600811
8	node1	1	1249598580
9	node1	1	1249598592
10	node2	1	1249598601
11	node3	1	1249598611
12	node6	1	1249598618
13	ndoe8	1	1249600191
14	node9	1	1249600819

Table 6.4 Ganglia Data in the Table *nodeshistory*

j_index	jobid	userid	grpid	jname	sessionid	rlimits	rused	queue	account	pro_time	ep_time
1	150.be	sas	rc	my_serial_job	150	0	node1	serial		1248733108	1248733108
2	151.bew	sas	rc	last_send	151	0	node7	serial		1248733120	1248733121
3	170.ho	aq7	daq	serial_1	170	0	node9	serial		1248733120	1248733120
4	151.gd	aq7	rc	par_job	10	0	node4	serial		1248733298	1248733302

Table 6.5 Torque Data in the Table *torque*

RAS Framework Engine Prototype

The package ras-0.4 is the latest fully tested version.
The package ras-0.5 is the latest development version.

6.3 Critical Review

There are many arguments about reliability analysis capability of making predictions. Some researchers are against the whole idea and some are eager to proceed further in research. In this case some questions need to be answered: Is there any evidence or notion of evidence that the failures are predictable and for how long they are predictable?

The answer is when looking at the system and the system is heating up, i.e. if the temperature is going up, the result would be the crash or explosion of the system. Looking at thresholds and tracking the incremental temperature it becomes clearly obvious that the temperature will exceed the thresholds. So the conclusion would be that some of the failures are predictable, especially such as hardware failures. More thoughts need to be given to software failures. Speaking about software failures: a software is considered to be ready for the production stage when there are no faults or at least no seen faults. But this does not mean that the software is free from faults.

The *ras* database was filling up very quickly, in 27 days the memory space shrank from 21 GB to 1.2 GB by gathering data at regular intervals of 30 seconds. Eventually, the collection was stopped as needed disk space was not available. This happened due to storing the all monitoring and syslog data without making decision on what exactly data need to be stored for analyses . But at this stage it is difficult to say what data is needed, only scientists who deal with data can decide on which data should be collected. In this sense some extra data can be added and avoided for the collection. Thus storing raw environmental monitoring data has a serious scalability challenge that will be considered in the future.

The Ganglia monitoring daemon (gmond) is running on all the nodes and broadcasts the metrics values to every node. This has a serious scalability problem that will be improved in the future through a scalable data aggregation and reduction approach. One of the approaches is to broadcast the collected data not to every gmond but to use a tree model, i.e. 2 gmond send to one gmond and so on until the one (gmond on a head node) has all collected data.

Maintaining global synchronized time in a massively distributed is not impossible, but comes at a cost, such as increased network traffic and OS noise, that is just too high for scientific HPC systems. Any analysis needs to take into account that the accuracy of timestamps in the database is not in microseconds or milliseconds but in seconds or tens of seconds. This problem needs to be considered as well.

The whole idea about RAS Framework Engine Prototype is not to convince that it is the only possible solution or the best solution of handling failures, but attempt is to provide or propose a method helping to improve the correct state of the practical art, or to put research in Proactive FT and get some experience in building a prototype considering that the Proactive FT might also be a good solution for handling failures. The designed, developed and deployed prototype for the RAS Engine was able to gather and store environmental system monitoring, system even logging, parallel job monitoring and system resource monitoring data to the DB. The overall requirements for the Prototype creation were satisfied.

6.4 Future Work

Future work will concentrated on building the Reliability Analysis for making predictions as this is a major part of the RAS Framework. The analysis of data in online mode, i.e. analyse raw data and store it to the database, is to be made later on. Later actions can be taken on stored data thanks to using the archiving approach. One of the possible solutions is to use Job Monitoring and Archiving system (Job Monarch) [15]. Job Monarch provides the ability to archive job statistics so that cluster users can find job information of old and maybe failed jobs and to analyse possible problems. Archiving saves memory space and data can be brought back from the archive for analysis.

Stored data in database can be reduced by statisticians, i.e. users working on the RA model. For example, instead of storing all information from Event monitoring system, store warnings and error messages only. And, as it was mentioned before, in the critical review, both time drifting and gmond scalability needs to be considered. The next step would be to test the RAS Framework Engine Prototype on a HPC system.

After the improvements are done it would be possible to proceed with the software implementation, operations and maintenance for the software and its disposition on the market.

6.5 Summary

This chapter describes achieved successes and limitations of the RAS Framework Engine Prototype. The requirements of the project are fulfilled and the future work is proposed.

7. REFERENCES

[1] Exploring Science Frontiers at Petascale. NCCS Reports. URL: http://www.nccs.gov/wp-content/media/nccs_reports/Petascale_Brochure.pdf

[2] TOP500 List - June 2009 (1-100). URL: http://www.top500.org/list/2009/06/100

[3] National Centre for Computational Sciences. URL: http://www.nccs.gov

[4] Christian Engelmann. Symmetric Active/Active High Availability for High-Performance Computing System Services. PhD thesis, Department of Computer Science, University of Reading, UK, 2008. Thesis research performed at Oak Ridge National Laboratory. Advisor: Prof. Vassil N. Alexandrov (University of Reading). URL: http://www.csm.ornl.gov/~engelman/publications/engelmann08symmetric3.pdf

[5] Christian Engelmann. High-Performance Computing Research at Oak Ridge National Laboratory. Invited talk at the Reading Annual Computational Science Workshop, Reading, United Kingdom, December 8, 2008. URL: http://www.csm.ornl.gov/~engelman/publications/engelmann08high.ppt.pdf

[6] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, Jose C. Sancho. Entering the Petaflop Era: The Architecture and Performance of Roadrunner. Year of Publication: 2008. Source: Conference on High Performance Networking and Computing. Proceedings of the 2008 ACM/IEEE conference on Supercomputing. Austin, Texas. Article No. 1. Publisher: IEEE Press Piscataway, NJ, USA. URL: http://portal.acm.org/citation.cfm?id=1413372

[7] Salim Hariri and Manish Parashar. Tools and Environments for Parallel and Distributed Computing. Wiley InterScience, John Wiley & Sons, Inc., Hoboken, NJ, USA, January 2004. ISBN 978-0-471-33288-6. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471332887.html

[8] Barney, Blaise. Introduction to Parallel Computing. Retrieved 2007-11-09 Lawrence Livermore National Laboratory. URL: https://computing.llnl.gov/tutorials/parallel_comp/

[9] David Peleg. Distributed Computing: A Locality-Sensitive Approach, 2000. SIAM, ISBN 0-89871-464-8. URL: http://www.ec-securehost.com/SIAM/DT05.html

[10] Sukumar Ghosh: Distributed Systems: An Algorithmic Approach, 2006 CRC Press (ISBN 1584885645).URL: http://search.barnesandnoble.com/Distributed-Systems/Sukumar-Ghosh/e/9781584885641

[11] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. December 18, 2006. Technical Report No. UCB/EECS-2006-183 EECS. Department University of California, Berkeley. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

[12] Michael T. Heath. Scientific Computing, 2nd Edition. August 2001. McGraw-Hill, New York, USA,. ISBN 978-0-072-39910-3. URL: http://catalogs.mhhe.com/mhhe/viewProductDetails.do?isbn=0072399104.

[13] Bianca Schroeder, Garth A. Gibson. A Large -scale study of Failures in High-Performance Computing Systems. Proceedings of the International Conference on Dependable Systems and Networks (DSN2006), Philadelphia, PA, USA, June 25-28, 2006. Supercedes Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-05-112, December, 2005.

[14] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive Process-Level Live Migration in HPC Environments. In Proceedings of the 21st IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2008, pages 1-12, Austin, TX, USA, November 15-21, 2008. ACM Press, New York, NY, USA. ISBN 978-1-4244-2835-9. URL: http://www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf

[15] Monitoring and Archiving system. URL https://subtrac.sara.nl/oss/jobmonarch

[16] Distributed Debugging Tool Training. Allinea Scale To New heights. Alliean Software 2009

[17] Franck Cappello, Laxmikant Kale, Frank Mueller, Keshav Pingali and Alexander Reinefeld. Fault Tolerance in High-Performance Computing and Grids. June 4_8, 2009, the Dagstuhl Seminar. Schloss Dagstuhl _ Leibniz Center for Informatics. URL: http://drops.dagstuhl.de/opus/volltexte/2009/2040/pdf/09191_abstracts_collection.2040.pdf

[18] Y. Li, Z. Lan, P. Gujrati, and X. Sun, "Fault-Aware Runtime Strategies for High Performance Computing", IEEE Trans. on Parallel and Distributed Systems , vol. 20(4), pp. 460-473, 2009. URL: http://www2.computer.org/portal/web/csdl/doi/10.1109/TPDS.2008.12

[19] Geoffroy Vallee, Kulathep Charoenpornwattana, Christian Engelmann, Anand Tikotekar, Chokchai “Box” Leangsuksun, Thomas Naughton, Stephen L. Scott. A Framework for Proactive Fault Tolerance. ARES, pp.659-664, 2008 Third International Conference on Availability, Reliability and Security, Barcelona, Spain. URL: http://www2.computer.org/portal/web/csdl/doi/10.1109/ARES.2008.171

[20] Reliability, Availability and Serviceability URL: http://whatis.techtarget.com/definition/0,,sid9_gci1232701,00.html

[21] Don Clausing, Daniel D. Frey. Improving system reliability by failure-mode avoidance including four concept design strategies.Pages 245-261, Vol. 8; issue 3, September 2005, SYSTEMS ENGINEERING -NEW YORK. URL: http://meche.mit.edu/documents/danfrey/danfrey_improving.pdf

[22] Christian Engelmann, Geoffroy R. Vallée, Thomas Naughton, and Stephen L. Scott. Proactive Fault Tolerance Using Preemptive Migration. In Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009, pages 252-257, Weimar, Germany, February 18-20, 2009. IEEE Computer Society, Los Alamitos, CA, USA. ISBN 978-0-7695-3544-9. ISSN 1066-6192. URL: http://www.csm.ornl.gov/~engelman/publications/engelmann09proactive.pdf

[23] Fault-Tolerant Definition.

URL: http://searchcio-midmarket.techtarget.com/sDefinition/0,,sid183_gci214456,00.html

[24] Stephen L. Scott, Christian Engelmann, Geoffroy R. Vallée, Thomas Naughton, Anand Tikotekar, George Ostrouchov, Chokchai (Box) Leangsuksun, Nichamon Naksinehaboon, Raja Nassar, Mihaela Paun, Frank Mueller, Chao Wang, Arun B. Nagarajan, and Jyothish Varma. A Tunable Holistic Resiliency Approach for High-Performance Computing Systems. Poster at the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) 2009, Raleigh, NC, USA, February 14-18, 2009. URL: http://www.csm.ornl.gov/~engelman/publications/scott09tunable.pdf

[25] A Paul Millar1, G A Stewart1, G A Cowan. Monitoring with MonAMI: a case study. Journal of Physics: Conference Series, Volume 119, Issue 6, pp. 062037 (2008). URL: http://www.gridpp.ac.uk/papers/Monami-paper-CHEP07.pdf

[26] Software Testing Techniques URL: http://www.his.sunderland.ac.uk/~cs0mel/comm83wk5.doc

[27] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, Stephen L. Scott. Proactive fault tolerance for HPC with Xen virtualization. International Conference on Supercomputing, 2007. New York, NY, USA ISBN:978-1-59593-768-1.

URL: ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/.../TR-2007-1.pdf

[28] Chao Wang, Frank Mueller, Christian Engelmann, Stephen L. Scott

Proactive process-level live migration in HPC environments.Conference on High Performance Networking and Computing.IEEE Press Piscataway, NJ, USA. Article No. 43. 2008 ISBN:978-1-4244-2835-9
URL://www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf

[29] Mean Time Between Failures.

URL: http://mccltd.net/blog/wp-content/uploads/2009/07/mtbf.jpg

[30] Reliability, Availability, and Serviceability (RAS) for Petascale High-End Computing and Beyond. URL:
http://images.google.com/imgres?imgurl=http://www.fastos.org/ras/images/framework-proactive-
big.png&imgrefurl=http://www.fastos.org/ras/&usg=__0BP_zFSzP5j5wAN4x1eIfcVZuIE=&h=338&w=590&sz=75&hl=en&start=1&um=1&tbnid=VI5XiNrN60wzhM:&tbnh=77&tbnw=135&prev=/images%3Fq%3Dras%2Bfault%2Btolerance%2Bmechanism%26hl%3Den%26client%3Dfirefox-a%26rls%3Dcom.ubuntu:en-GB:unofficial%26sa%3DN%26um%3D1

[31] High Availability for High-End scientific computing, Master's thesis, Kai Uhlemann 2006. University of Reading.

URL: www.csm.ornl.gov/~engelman/publications/engelmann05high.pdf

[32] Torque Resource Manager.

URL: http://www.clusterresources.com/torquedocs21/a.gprologueepilogue.shtml

[33] Centralised syslog-ng. URL: http://www.vermeer.org:

[34] Torque Resource Manager. URL: http://www.clusterresources.com/products/torque-resource-manager.php

[35] Moab Cluster Suite. URL: http://www.clusterresources.com/products/moab-cluster-suite.php/

[36] Syslog daemon. URL: http://linux.about.com/od/commands/l/blcmdl8_syslogd.htm

[37] Kernel logging daemon. URL: http://linux.about.com/cs/linux101/g/klogd.htm

[38] Centralised Syslog-ng. URL: http://linux.about.com/cs/linux101/g/syslogng.htm

[39] Ganglia Monitoring System. URL: http://ganglia.info/

[40] Nagios - IT Infrastructure Monitoring URL: http://www.nagios.org/about/features

[41] A comparison of Oracle and MySQL, Greff Petri, 2005

[42] Sayantan Chakravorty, Celso L. Mendes, and Laxmikant V. Kale. Proactive fault tolerance in MPI Application Via Task Migration. HiPC 2006, LNCS 4297, pp. 485-496,2006. Springer-Verlag Berlin Heidelberg 2006

[43] Jim Brandt, Bert Debusschere, Ann Gentile, Jackson Mayo, Philippe P?bay, David Thompson, Matthew Wong, "Using Probabilistic Characterization to Reduce Runtime Faults in HPC Systems," ccgrid, pp.759-764, 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008

[44] OVIS. URL: http://ovis.ca.sandia.gov

[45] Geoffroy Vallee, Kulathep Charoenpornwattana, Christian Engelmann, Anand Tikotekar, Chokchai Leangsuksun, Thomas Naughton, Stephen L. Scott, "A Framework for Proactive Fault Tolerance," ares, pp.659-664, 2008 Third International Conference on Availability, Reliability and Security, 2008. URL: http://www2.computer.org/portal/web/csdl/doi/10.1109/ARES.2008.171

[46] J.T. Daly, “Methodology and Metrics for Quantifying Application Throughput,” Proceedings of the Nuclear Explosives Code Developers’ Conference (NECDC), in press.

[47] Bianca Schroeder, Garth A. Gibson. A large-scale study of failures in high-performance computing systems. Proceedings of the International Conference on Dependable Systems and Networks (DSN2006), Philadelphia, PA, USA, June 25-28, 2006. Supercedes Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-05-112, December, 2005.

[48] The raw data. URLs: http://www.pdf.cmu.edu/FailureData

[49] Yawei Li, Zhiling Lan, Prashasta Gujrati, Xian-He Sun, "Fault-Aware Runtime Strategies for High-Performance Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 4, pp. 460-473, Apr. 2009, doi:10.1109/TPDS.2008.128 URL: http://www2.computer.org/portal/web/csdl/doi/10.1109/TPDS.2008.128

[50] A. Oliner, R. Sahoo, J. Moreira, and M. Gupta. Gault -Aware Job Scheduling for BlueGene/L Systems, Proc 18th Int'l Parallel and Distributed Processing Symp. (IPDPS '04), p.64, 2004.

[51] Yuyan Liu, Wooi Ping Cheah, Byung-Ki Kim, Hyukro Park.Predict Software Failure-prone by Learning Bayesian Network. International journal of advance science and technology.

[52] R. Gupta, P. Beckman, B. H. Park, E. lusk, P. Hargrove, A. Geist, D. K. Panda, A. Lumsdaine and J. Dongara. CIFTS: A coordinated infrastructure for fault-tolerant systems.

URL: http://www.mcs.anl.gov/research/cifts/

[53] Mootaz Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David B. Johnson. A survey of rollback-recovery protocols in Message- Passing Systems. NY, USA. Volume 34 , Issue 3 (September 2002) Pages: 375 - 408   ISSN:0360-0300 URL: http://portal.acm.org/citation.cfm?id=568522.568525

[54] A. Borg, W. Blau, W. Graetsch, F. Hermann. and W. Oberle. Fault tolerance under Unix. ACM transactions on computing systems, 7(1):1-24, Feb 1989

[55] J. T. Daly, L. A. Pritchett-Sheats, S. E. Michalak, "Application MTTFE vs. Platform MTBF: A Fresh Perspective on System Reliability and Application Throughput for Computations at Scale," ccgrid, pp.795-800, 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008.

URL: http://www2.computer.org/portal/web/csdl/doi/10.1109/CCGRID.2008.103

[56] Jon Stearley, Adam J. Oliner. Bad Words: Finding Faults In Spirit's Syslogs. Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on. 19-22 May 2008 Page(s): 765-770 Lyon, ISBN: 978-0-7695-3156-4

URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4534301

[57] William M. Jones, John T. Daly and Nathan A. DeBardeleben. Application Resilience: Making progress In Sprite of Failures. Workshop on Resiliency in High Performance Computing (RESILIENCE 2008)

[58] Jyothish Varma, Chao Wang, Frank Mueller, Christian Engelmann, Stephen L. Scott. Scalable,fault-tolerant membership for MPI Tasks on HPC Systems. Proceedings of the 20th annual international conference on Supercomputing, Cairns, Queensland, Australia, 2006, ISBN:1-59593-282-8 . URL: http://portal.acm.org/citation.cfm?id=1183401.1183433

[59] Kulathep Charoenpornwattana, Chokchai Leangsuksun, Geoffroy Vallee, Anand Tikotekar, Stephen L. Scott. A neural Networks Approach for Intelligent Fault prediction in HPC Environments. 2008. URL: xcr.cenit.latech.edu/hapcw2008/program/papers/101.pdf

[60] Song Fu, Cheng-Zhong Xu, Exploring Event Correlation for Failure Prediction in Coalitions of Clusters. Proceedings of the 2007 ACM/IEEE conference on Supercomputing. Reno, Nevada Article No. 41, 2007, ISBN:978-1-59593-764-3

URL: http://portal.acm.org/citation.cfm?id=1362678

[61] Sayantan Chakravorty, Celso L. Mendes, Laxmikant V. Kale. Proactive Fault Tolerance in Large Systems. URL: charm.cs.uiuc.edu/papers/EvacuationHPCRI05.pdf

[62] Lawrence Berkeley National Laboratory, Berkeley, CA, USA. Berkeley Lab Checkpoint/Restart (BLCR) documentation, 2007. URL http://ftg.lbl.gov/checkpoint.

[63] Indiana University, Bloomington, IN, USA. Local Area Multicomputer Message Passing Interface (LAM-MPI) documentation, 2007. URL http://www.lam-mpi.org.

[64] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack J. Dongarra. MPI: The Complete Reference (Vol. 1), 2nd Edition. MIT Press, Cambridge, MA, USA, September 1998. ISBN 978-0-262-69215-1.

URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=3898.

[65] Pacific Northwest National Laboratory, Richland, WA, USA. TICK: Transparent Incremental Checkpointing at Kernel-level documentation, 2007.

URL http://hpc.pnl.gov/sft/tick.html.

[66] James S. Plank, Kai Li, and Michael A. Puening. Diskless checkpointing. IEEE Transactions on Parallel and Distributed Systems (TPDS), 9(10):972–986, 1998.IEEE Computer Society. ISSN 1045-9219.

URL http://doi.ieeecomputersociety.org/10.1109/71.730527.

[67] George Bosilca, Remi Delmas, Jack Dongarra and Julien Langou. Algorithmic Based Fault Tolerance Applied to High performance Computing. Journal of Parallel and Distributed Computing, 69(4):410 - 416., MIMS Eprint: 2009.3, January 2009,

URL: http://www.sciencedirect.com/science/article/B6WKJ-4V8GB44-2/2/2658d9756341bece20e06d1485456678

[68] Innovative Computing Laboratory (ICL), Computer Science Department, University of Tennessee, Knoxville, TN, USA. FT-MPI documentation, 2007.

URL: http://icl.cs.utk.edu/ftmpi.

[69] Open MPI Team. Open MPI documentation, 2007. URL http://www.open-mpi.org.

[70] MPICH-V. URL: http://mpich-v.lri.fr/index.php?section=intro&subsection=intro

[71] Pablo Neira Ayuso, Laurent Lefevre, Rafael Martinez Gasca, "hFT-FW: Hybrid Fault-Tolerance for Cluster-Based Stateful Firewalls," icpads, pp.525-532, 2008 14th IEEE International Conference on Parallel and Distributed Systems, 2008

[72] Top 500 Supercomputers. URL: http://www.top500.org/list/2003/11/

[73] Extended System of Units. URL: http://www.mindspring.com/~jimvb/unitsystem.htm

[74] XEN hypervisor. URL: http://www.xen.org/

[75] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, Stephen L. Scott. Proactive fault tolerance for HPC with Xen virtualization. International Conference on Supercomputing. 2007 ISBN:978-1-59593-768-1.

URL: ftp://ftp.csc.ncsu.edu/pub/tech/2007/TR-2007-1.pdf

[76] Chao Wang, Grank Mueller, Christian Englemann, Stephen L. Scott. Proactive process-level live migration in HPC environments. Conference on High Performance Networking and Computing. 2008 ISBN:978-1-4244-2835-9.

URL: www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf

[77] Emmanuel Aviles, Shamir J. Quinones, Yael M. Camacho. Detecting Anomalies for High Performance Computing Resilience, Poster 2009 ORNL

[78] Ron I. Resnick. A modern taxonomy of high availability. URL: http://www.verber.com/mark/cs/systems/A%20Modern%20Taxonomy%20of%

20High%20Availability.htm, 1996.2005.

[79]Portable Batch System URL: http://www.pbsgridworks.com/

[80] Portable Batch System URL: http://hpc.sissa.it/pbs/pbs-1.html#ss1.1

[81] Maui Cluster Scheduler URL: http://www.clusterresources.com/maui

[82] High-Availability.com, K. UK. RSF-1 documentation, 2007 URL: https://www.high-availability.com/links/2-8-rsfl.php.

[83] OpenIPMI URL http://openipmi.sourceforge.net

[84] ML Massie, BN Chun, DE Culler. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, 2004

URL: ganglia.info/papers/science.pdf

[85] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pebay, D. Thompson, and M. Wong. OVIS-2 : A robust distrubuted architecture for scalable RAS. In IEEE International Parallel and Disributed Processing Sysmposium: wokshop on System Mangement Techniques, Processes, and Services, 2008.URL https://ovis.ca.sandria.gov/mediawiki/images/6/60/OVIS-ipdps08.pdf

[86] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3):382401, 1982. ACM Press, New York, NY, USA. ISSN 0164-0925. URL http://doi.acm.org/10.1145/357172.357176.

8. APPENDICES

8.1 Abbreviations

RAS	Reliability Availability and Serviceability
ORNL	Oak Ridge National Laboratory
LANL	Los Alamos National Laboratory
PF	petaflops
TF	teraflops
UT	University of Tennessee
XML	External Mark-up Language
XSL	External Stalesheet Language
XSLT	External Stalesheet Language Transformation
HPC	High Performance Computing
OS	Operating System
SQL	Structured Query Language
DB	Database
DBMS	Database Management System
TN	number of seconds since the metric was last updated
TMAX	freshness of a metric
DMAX	number of seconds ganglia retains an old metric
gmond	ganglia monitoring daemon
gmetric	ganglia metric
FT Fault	Tolerance
FZJ	Forschungszentrum Juelich
LLNL	Lawrence Livermore National Laboratory
MTTI	Mean Time To Interrupt
MPI	Message Passing Interface
OpenMPI	Message Passing Interface
I/O	Input/Output
FS	File System
HEC	High-End Computing
LAN	Local Area Network
HDD	Hard Disk Drive
AMTTF	Application Mean Time To Failure
SMTTF	System Mean Time To Failure
AMPI	Adaptive Message Passing Interface
MTTF	Mean Time To Failure
MTBR	Mean Time Between Error
BN	Bayesian Network
MTTR	Mean Time To Repair
FARS	Fault-Aware Runtime System
VM	Virtual Machine
VE	Virtual Environment
VMM	Virtual Machine Monitor
CPU	Central Processing Unit
PC	Personal Computer
PFT	Proactive Fault Tolerance
BLCR	Berkeley Laboratory Checkpoint Restart
LAM	Local Area Multicomputer
TICK	Transparent Incremental Checkpoiting at Kernel-level
TCP	Transmission Control Protocol

API	Application Programming Interface
RA	Reliability Analysis
RM	Resource Manager
JS	Job Scheduler
RE	Runtime Environment
PBS	Portable Batch System
XDR	XML Data Reduction
RRD	Round Robin Database
DDT	Distributed Debugging Tool

8.2 Complete Listings of RAS Framework Engine Prototype package ras-0.4

1. install.sh (ras)

2. uninstall.sh (ras)

3. install.sh (database)

4. uninstall.sh (database)

5. gangliamysqld.xsl (ganliamysqld)

6. gangliamysqld-init.sh (ganliamysqld)

7. gangliamysqld-run.sh (ganliamysqld)

8. install.sh (ganliamysqld)

9. unistall.sh (ganliamysqld)

10. syslogmysqld.conf (syslogmysqld)

11.syslogmysqld-init.sh (syslogmysqld)

12. syslogmysqld-run.sh (syslogmysqld)

13. install.sh (syslogmysqld)

14. uninstall.sh (syslogmysqld)

15. prologue.sh (torquemysql)

16. epilogue.sh (torquemysql)

17. install.sh (torquemysql)

18. uninstall.sh (torquemysql)

19. migrationd-init.sh (migrationd)

20. migrationd-run.sh (migrationd)

21. install.sh (migrationd)

22. uninstall.sh (migrationd)

8.3 User's Manual

ras-0.4 is the latest fully tested version and ras-0.5 is the latest development version, i.e. not tested.
This guide presumes that below mentioned software installed and running:
Ganglia (gmond)
Torque
Syslog-ng (2 other packeges: klogd, sysklogd will be removed):
sudo apt-get install syslog-ng

Knowledge of SQL and Torque commands are needed.

From README:

```
#!/bin/more

###################################################################################
#
# RAS framework.

# Copyright (c) 2009 Oak Ridge National Laboratory.
#
###################################################################################

Installation/Uninstallation

--------------------------

The installation of framework components is always on the local host the user is

logged in and requires super user privileges, e.g., via sudo. Components may be

installed on different hosts. In this case, a remote Database configuration is

installed. The provided package offers the folowing (un)installation scripts:

sudo ./install.sh              Installs the entire framework on localhost.

sudo ./uninstall.sh            Uninstalls the entire framework from localhost.

cd <component> && sudo ./install.sh

                        Installs a framework component on localhost.

                        If the Database component is not installed, a

                        remote Database configuration is installed.

cd <component> && sudo /uninstall.sh

                        Uninstalls a framework component from localhost.

                        If the Database component is not installed, a
```

remote Database configuration is uninstalled

if it is not needed by other components.

It is highly encouraged to install all components on the same host to ensure maximum performance. In some settings, however, a separate server for one or more components may exist.

For example, installing the Database component on node1, the Installing Ganglia/ MySQL Daemon component on node2, the Syslog/MySQL Daemon component on node3, the Torque/MySQL Scripts component on node4 and the Migration Daemon component on node5 involves the following steps:

1. Copy the RAS framwork package to all nodes:

```
scp -r ras-0.4 user@node1:/home/user
scp -r ras-0.4 user@node2:/home/user
scp -r ras-0.4 user@node3:/home/user
scp -r ras-0.4 user@node4:/home/user
scp -r ras-0.4 user@node5:/home/user
```

2. Install the Database component on node1:

```
ssh user@node1
     cd ras-0.4/datadase/
sudo ./install.sh
     exit
```

3. Install the Ganglia/MySQL Daemon component on node2 (with node1 as remote database):

```
ssh user@node2
        cd ras-0.4/gangliamysqld/
sudo ./install.sh
        exit
```

3. Install the Syslog/MySQL Daemon component on node3 (with node1 as remote database):

```
ssh user@node3
        cd ras-0.4/syslogmysqld/
sudo ./install.sh
        exit
```

4. Install the Torque/MySQL Scripts component on node4 (with node1 as remote database):

```
ssh user@node4
        cd ras-0.4/torquemysql/
sudo ./install.sh
        exit
```

4. Install the Migration Daemon component on node5 (with node1 as remote database):

```
ssh user@node5
        cd ras-0.4/migrationd/
sudo ./install.sh
        exit
```

Database Access

By using the MySQL command line interface:

mysql -u rasro ras	Enters the MySQL command line interface as user "rasro" (default read-only user name used here) using the "ras" database (default database name used here).
SELECT * from nodes;	From within the MySQL command line interface, queries the nodes table stored by the Ganglia/MySQL Daemon.
SELECT * from ganglia;	From within the MySQL command line interface, queries the ganglia table stored by the Ganglia/MySQL Daemon.
SELECT * from syslog;	From within the MySQL command line interface, queries the syslog table stored by the Syslog/MySQL Daemon.
SELECT * from torque;	From within the MySQL command line interface, queries the syslog table stored by the Torque/MySQL Scripts.
quit	Exits the MySQL command line interface.

Alternatively from a shell:

```
echo "SELECT * from nodes;"   | mysql -u rasro ras
echo "SELECT * from ganglia;" | mysql -u rasro ras
echo "SELECT * from syslog;"  | mysql -u rasro ras
echo "SELECT * from torque;"  | mysql -u rasro ras
```

8.4 Test Suite for RAS 0.4 Package

DATABASE

1. If not 'root', expected result the DB cannot be installed.

```
./install.sh
ERROR: Only root can install.
```

2. If 'root', expected result the DB is installed.

```
sudo ./install.sh
Querying Database configuration...
Database host                  : localhost
Database name           [ras]   : ras
Database read-write user [rasrw] : rasrw
Database read-only user  [rasro] : rasro
Is this an OSCAR cluster [y]     : n
Setting Database configuration defaults...
Installing Database configuration...
Installing Database...
Enter password:
```

3. If 'root' and a DB exists, expected result another 'ras' DB cannot be installed.

```
sudo ./install.sh
Querying Database configuration...
Database host                  : localhost
Database name           [ras]   : ras
Database read-write user [rasrw] : rasrw
Database read-only user  [rasro] : rasro
Is this an OSCAR cluster [y]     : n
Setting Database configuration defaults...
Installing Database configuration...
Installing Database...
Enter password:
ERROR 1007 (HY000) at line 1: Can't create database 'ras'; database exists
```

4. If not 'root', expected result the DB cannot be uninstalled.

```
./uninstall.sh

ERROR: Only root can uninstall.
```

5. If 'root' password for the DB is incorrect, expected result the DB cannot be uninstalled.

```
sudo ./uninstall.sh
Uninstalling MySQL database...
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
```

6. If 'root', expected result the DB can be unistalled.

```
sudo ./uninstall.sh
Uninstalling MySQL database...
Enter password:
Uninstalling Database configuration...
```

7. Other components based on the same principle, so tested for installation and unistallation.

GANGLIAMYSQLD

```
sudo ./install.sh
Querying Ganglia/MySQL Daemon configuration...
Ganglia host          [localhost] :
Ganglia port          [8649]      :
Ganglia/MySQL interval [30]       :
Setting Ganglia/MySQL Daemon configuration defaults...
Installing Ganglia/MySQL Daemon configuration...
Installing Ganglia/MySQL Daemon...
 Adding system startup for /etc/init.d/gangliamysqld ...
   /etc/rc0.d/K98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc1.d/K98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc6.d/K98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc2.d/S98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc3.d/S98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc4.d/S98gangliamysqld -> ../init.d/gangliamysqld
   /etc/rc5.d/S98gangliamysqld -> ../init.d/gangliamysqld
Starting Ganglia/MySQL Daemon...
Starting RAS Ganglia/MySQL Daemon: gangliamysqld.

sudo ./uninstall.sh

Stopping Ganglia/MySQL Daemon...
Stopping RAS Ganglia/MySQL Daemon: gangliamysqld.
Uninstalling Ganglia/MySQL Daemon...
 Removing any system startup links for /etc/init.d/gangliamysqld ...
   /etc/rc0.d/K98gangliamysqld
   /etc/rc1.d/K98gangliamysqld
   /etc/rc2.d/S98gangliamysqld
   /etc/rc3.d/S98gangliamysqld
   /etc/rc4.d/S98gangliamysqld
   /etc/rc5.d/S98gangliamysqld
   /etc/rc6.d/K98gangliamysqld
Uninstalling Ganglia/MySQL Daemon configuration...
```

SYSLOG-NG

```
sudo ./install.sh
Querying Syslog/MySQL Daemon configuration...
Syslog/MySQL interval [30] :
Setting Syslog/MySQL Daemon configuration defaults...
Installing Syslog/MySQL Daemon configuration...
Installing Syslog/MySQL Daemon...
 Adding system startup for /etc/init.d/syslogmysqld ...
   /etc/rc0.d/K98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc1.d/K98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc6.d/K98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc2.d/S98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc3.d/S98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc4.d/S98syslogmysqld -> ../init.d/syslogmysqld
   /etc/rc5.d/S98syslogmysqld -> ../init.d/syslogmysqld
Reconfiguring Syslog Daemon...
 * Stopping system logging syslog-ng                              [ OK ]
 * Starting system logging syslog-ng                              [ OK ]
Starting Syslog/MySQL Daemon...
Starting RAS Syslog/MySQL Daemon: syslogmysqld.

sudo ./uninstall.sh

Stopping Syslog/MySQL Daemon...
Stopping RAS Syslog/MySQL Daemon: syslogmysqld.
Reconfiguring Syslog Daemon...
 * Stopping system logging syslog-ng                              [ OK ]
 * Starting system logging syslog-ng                              [ OK ]
Uninstalling Syslog/MySQL Daemon...
 Removing any system startup links for /etc/init.d/syslogmysqld ...
   /etc/rc0.d/K98syslogmysqld
   /etc/rc1.d/K98syslogmysqld
   /etc/rc2.d/S98syslogmysqld
   /etc/rc3.d/S98syslogmysqld
   /etc/rc4.d/S98syslogmysqld
   /etc/rc5.d/S98syslogmysqld
   /etc/rc6.d/K98syslogmysqld
Uninstalling Syslog/MySQL Daemon configuration...
```

TORQUEMYSQL

```
sudo ./install.sh
Querying Torque/MySQL Scripts configuration...
TORQUE_HOME   [/var/spool/torque/] :
Setting Torque/MySQL Scripts configuration defaults...
Installing Torque/MySQL Scripts configuration...
Installing Torque/MySQL Scripts...
```

sudo ./uninstall.sh
Uninstalling Torque/MySQL Scripts...
Uninstalling Torque/MySQL configuration...

8. Package ras-0.4 is intalled and uninstalled successfully.

9. gangliamysqld.xsl (ganliamysqld). Expected all fields to be inserted. The result is successful.

INSERT INTO `ganglia` (`sequence`,` name`, `value`, `type`, `units`, `tn`, `tmax`, `dmax`, `slope`, `source`, `node`, `start`, `report`) VALUES
('null','disk_total','39.279','double','GB','37','1200','0','both','gmond','node6.xtorc','1245777829','1248733108'),
('null','cpu_speed','1995','uint32','MHz','37','1200','0','zero','gmond','node6.xtorc','1245777829','1248733108',);

INSERT INTO `nodesstate` (`nodename`, `start`, `updated`) VALUES
('node1', 1249600000, 1249600819),
('node2', 1249600000, 1249601047);

10.gangliamysqld-run.sh (ganliamysqld). Expected all fields to be inserted. The result is successful.

INSERT INTO `nodeshistory` (`sequence`, `nodename`, `status`, `timestamp`) VALUES
(1, 'node1', 1, 1249600811),
(2, 'node5', 1, 1249600811);

11.syslogmysqld.conf (syslogmysqld). Expected all fields to be inserted. The result is successful.

INSERT INTO `syslog` (`seq`, `host`, `facility`, `priority`, `level`, `tag`, `reported`, `recorded`, `program`, `msg`) VALUES
(1, 'ubuntu', 'syslog', 'notice', 'notice', '2d', '1249600010', '1249600011', 'syslog-ng', 'syslog-ng[15156]: syslog-ng starting up; version="2.0.9"'),
(2, 'ubuntu', 'user', 'info', 'info', '0e', '1249600011', '1249699912', '/usr/sbin/gmeta', '/usr/sbin/gmetad[5464]: data_thread() got not answer from any [my cluster] datasource'),
(3, 'ubuntu', 'authpriv', 'info', 'info', '56', '1249600013', '1249600014', 'CRON', 'CRON[15169]: pam_unix(cron:session): session opened for user root by (uid=0)');

12. prologue.sh (torquemysql).Expected all fields to be inserted. The result is successful.

INSERT INTO `torque` (`j_index`, `jobid`, `userid`, `grpid`, `jname`, `sessionid`, `rlimits`, `rused`, `queue`, `account`, `pro_time`, `ep_time`) VALUES
(1, '150.be', 'sas', 'rc', 'my_serial_job', 150, '0', 'node1', 'serial', '', '1247574895', '1247574895'),
(2, '151.bew', 'sas', 'rc', 'last_send', 151, '0', 'node7', 'serial', '', '1247574895 ', '1247574895'),
(3, '170.ho', 'aq7', 'daq', 'serial_1', 170, '0', 'node9', 'serial', '', '1247574896', '1247574899'),
(4, '151.gd', 'aq7', 'rc', 'par_job', 10, '0', 'node4', 'serial', '', '1247574897', '1247574899');

13. epilogue.shepilogue.sh (torquemysql). Expected all fields to be inserted. The result is successful.

UPDATE `torque` SET (`j_index`, `jobid`, `userid`, `grpid`, `jname`, `rlimits`, `rused`, `queue`, `account`, `pro_time`, `ep_time`) VALUES
(1, '150.be', 'sas', 'rc', 'my_serial_job', '0', 'node1', 'serial', '1247574895'),

```
(2, '151.bew', 'sas', 'rc', 'last_send', '0', 'node7', 'serial', '1247574895 ');

UPDATE `torque` SET  `sessionid`, `rused`,  `ep_time` WHERE '170.ho' and
pro_time<='1247574896';

UPDATE `torque` SET `sessionid`, `rused`,  `ep_time`) WHERE '151.bew' and pro_time<=
'1247574895 ';
```

8.5 Software Requirements Specification (SRS) for RAS FRAMEWORK.

8.5.1 Introduction

Purpose

It is an open source software (steal under development), RAS framework ras-0.4.tar.gz (release in 2009) that deals with failures proactively in HPC. This document describes requirements and specifications for the system.
The scope of the product is a data storage, a prediction and a migration.

Document Conventions

Installation have some priorities, but each peace can be installed separately and order do not matter accept for database and migration. It must be installed first and migration daemon the last.

Intended Audience and Reading Suggestions

This document was designed as clear as possible but some computer background will help to understand the concepts in the document. This document is for every user: developers, testers who will improve the implementation, project managers, marketing staff, users and documentation writers and to those who will find this document useful. The rest contains of overall description, system features, external interface requirements and other non-functional requirements. If the reader is familiar with the chapter he can skip it and continue reading other chapters.

Project Scope

RAS Framework ras-0.1.tar.gz is a system to deal with failures in HPC systems. Its objectives and goal are to transfer aggregated data from different sources and store them to a database. Upon basic predictions a pre-emptive migration is taking place. (More information can be found in the Introduction chapter).

References

This document refers to Proactive Fault Tolerance with pre-emptive migration (*http://portal.acm.org/citation.cfm?id=1548888.1549715*).
More information can be provided about the next release(in 2010) by contacting *a.litvinova@student.reading.ac.uk*, *engelmannc@ornl.gov*.

Details about other softwares what were used can be found on these websites:
MySQL Database Management System: (http://www.mysql.com/)
Ganglia Monitoring System (http://ganglia.info/)
Torque Resource Management: (http://www.clusterresources.com/torque/)
Syslog-ng Logging System: (http://www.balabit.com/network-security/syslog-ng/)

8.5.2 Overall Description

Product Perspective

RAS Framework is a first baggy version. It relies on other softwares:
MySQL database management system, Ganglia monitoring system, Torque resource manager, syslog-ng Logging system. It is assumed that these software already installed and configured before RAS Framework installation. OSCAR can be used for faster installation on a cluster, and OSCAR-V on a virtual cluster. Interfaces between RAS Framework and these softwares are included in RAS Framework.

Product Features

Functions are: storing data, data analysis, migration, archiving. A user can query a database content. Details provided in Section 3.

Operating Environment

Hardware requirements: x86 ..., Virtual cluster must supports PVM, processors.
Linux Operating System: Ubuntu 8.10, Kubuntu 9.04 version ...
(Tested on) Cluster of 1-48 nodes, virtual cluster with physical 1-5 nodes.

Design and Implementation Constraints

Limitations to be improved by developers:
Reliability Analysis, Migrations Mechanism, Archiving, use of Checkpoint/restart, Database design.
Requirements: support PVM, high memory requirements as 100GB and higher, MySQL database to be used, MPI for parallel computing.
Languages: SQL, XSL, shell script.
Security: only a user with 'root' privileges can install (with sudo function).
TCP/IP protocols.

User Documentation

User documentations are: User's manual, maintenance manual, CD with a copy of the RAS Framework software. This information can be find in Appendices.
To get help or report a bag contact: *a.litvinova@student.reading.ac.uk*, *engelmannc@ornl.gov*.

Assumptions and Dependencies

RAS Framework is open sources software with all right reserved and uses other open sources softwares: Ganlia, Torque, MySQL and Syslog-ng with all right reserved.

8.5.3 System Features

This section illustrates the functional requirements for the product by system features, the major services provided for the product. More details can be found in Preliminary System design chapter.

RAS Framework System

RAS Framework:
install.sh, uninstall.sh, README.

1 Description and Priority

It is a central component of failure handling in HPC systems.

2 Stimulus/Response Sequences

Installing the cetral component will interact with a user in the console by asking to provide some information, some default can be used. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1: sudo user
REQ-2: data input
REQ-3: MySQL
REQ-4: Ganglia gmond
REQ-5: Torque
REQ-6: Syslog-ng

Database Component

Database creation:
database (install.sh, uninstall.sh)

1 Description and Priority

Creating of ras database using MySQL DBMS, with tables, metadata and users with privileges.

2 Stimulus/Response Sequences

Installing database component the installation process will interact with a user in the console by asking to provide database name, users names, some default can be used. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1:sudo user
REQ-2:data input
REQ-3: MySQL

Ganglia MySQL Daemon Component

Ganglia MySQL daemon:
gangliamysql (gangliamysqld.xsl, gangliamysqld-init.sh, gangliamysqld-run.sh,install.sh, unintsall.sh.)

1 Description and Priority

Creating interface between Ganglia and MySQL to store aggregated data from Ganglia to ras database. Interval for how often the data is stored can be adjusted in gangliamysqld-run.sh script.

2 Stimulus/Response Sequences

Installing interface ganglia mysql daemon component ,the installation process will interact with a user in the console. Database must be created before installing ganliga mysql daemon. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1:sudo user
REQ-2:Ganglia
REQ-3: MySQL
REQ-4: ras Database created

Torque MySQL Scripts Component

Torque MySQL scripts:
torquemysql (prologue.sh, epilogue.sh, install.sh, uninstall.sh)

1 Description and Priority

Creating interface between Torque and MySQL to store data about submited jobs from Torque to ras database.

2 Stimulus/Response Sequences

Installing interface torque mysql scripts. Database must be created before installing torque mysql scripts. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1:sudo user
REQ-2:Torque
REQ-3: MySQL
REQ-4: ras database created

Syslog MySQL Daemon Component

Syslog mysql daemon:
syslogmysqld (syslogmysqld.conf, syslogmysqld-init.sh, syslogmysqld-run.sh, install.sh, uninstall.sh)

1 Description and Priority

Creating interface between Syslog-ng and MySQL to store log massages from Syslong-ng to ras database. Interval for how often the data is stored can be adjusted in syslogmysqld-run.sh script. Souces, priorities (e.t. only warning and error messages) can be adjusted in syslogmysqld.conf.

2 Stimulus/Response Sequences

Installing interface syslog mysql daemon component. Database must be created before installing ganliga mysql daemon. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1:sudo user
REQ-2:Syslog-ng
REQ-3: MySQL
REQ-4: ras database created

Migration Daemon Component

migration daemon:
migrationd(migrationd-init.sh, migrationd-run.sh, install.sh,uninstall.sh)

1 Description and Priority

Migration mechanism with data analys.

2 Stimulus/Response Sequences

Installing interface migration daemon component . All other components must be installed and function in order to make a decision and triger the migration. The process of installation can be seen in the console.

3 Functional Requirements

Only user with sudo can install. On an error input the next step is suggested for further actions.

REQ-1:sudo user
REQ-2: MySQL
REQ-3:Ganglia gmond
REQ-4:Torque
REQ-5: Syslon-ng
REQ-6: ras database created

8.5.4 External Interface Requirements

User Interfaces

To work on Reliability Analysis model user is able to query ras database content by SELECT function, convert time from unix to human readable time (year.month.day hour.minute.second) with FROM_UNIX() function and to get data about the node status either it is up or down.
User interacts through Linux Terminal, no GUI is available.
PHPMyAdmin software can be installled for graphical interaction with ras Database.
Text Editors as vi, gedit, kate can be installed and used to modify configuration files.

Communications Interfaces

Web-browser can be used for PHPMyAdmin.
TCP/IP protocols.
SSH can be used to access a cluster remotely.
scp – secure copy transfer.

8.5.5 Other Non-functional Requirements

Safety Requirements

User with sudo privileges must know exactly what he/she is doing otherwise the user can corrupt the system/cluster.

Security Requirements

Two users are created with the installation:
rasrw read write user. The user can select, update, delete, insert data.
rasro read-only user. The user can select only.

Software Quality Attributes

The aim of this framework is to provide reliability, availability and serviceability of a system/cluster.
Reliability: avoid predicted failures
Availability: MTBF increase.
Serviceability: all existing services on the system/cluster are available.

8.5.6 Other Requirements

All rights reserved.

8.6 Ganglia Metrics

Ganglia metrics

load_one	One minute load average (module load_module)
os_release	Operating system release date (module sys_module)
mem_total	Total amount of memory displayed in KBs (module mem_module)
cpu_intr	cpu_intr (module cpu_module)
proc_run	Total number of running processes (module proc_module)
load_five	Five minute load average (module load_module)
disk_free	Total free disk space (module disk_module)
gexec	gexec available (module core_metrics)
mem_cached	Amount of cached memory (module mem_module)
mtu	Network maximum transmission unit (module sys_module)
cpu_sintr	cpu_sintr (module cpu_module)
pkts_in	Packets in per second (module net_module)
location	Location of the machine (module core_metrics)
bytes_in	Number of bytes in per second (module net_module)
swap_total	Total amount of swap space displayed in KBs (module mem_module)
bytes_out	Number of bytes out per second (module net_module)
load_fifteen	Fifteen minute load average (module load_module)
mem_free	Amount of available memory (module mem_module)
os_name	Operating system name (module sys_module)
boottime	The last time that the system was started (module sys_module)
cpu_idle	Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request (module cpu_module)
cpu_aidle	Percent of time since boot idle CPU (module cpu_module)
cpu_nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority (module cpu_module)
cpu_user	Percentage of CPU utilization that occurred while executing at the user level (module cpu_module)
sys_clock	Time as reported by the system clock (module sys_module)
mem_buffers	Amount of buffered memory (module mem_module)
cpu_system	Percentage of CPU utilization that occurred while executing at the system level (module cpu_module)
part_max_used	Maximum percent used for all partitions (module disk_module)
disk_total	Total available disk space (module disk_module)
heartbeat	Last heartbeat (module core_metrics)
mem_shared	Amount of shared memory (module mem_module)
machine_type	System architecture (module sys_module)
cpu_wio	Percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request (module cpu_module)
cpu_num	Total number of CPUs (module cpu_module)
proc_total	Total number of processes (module proc_module)
cpu_speed	CPU Speed in terms of MHz (module cpu_module)
pkts_out	Packets out per second (module net_module)
swap_free	Amount of available swap memory (module mem_module)

8.7 XML Example

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE GANGLIA_XML [
  <!ELEMENT GANGLIA_XML (GRID|CLUSTER|HOST)*>
    <!ATTLIST GANGLIA_XML VERSION CDATA #REQUIRED>
    <!ATTLIST GANGLIA_XML SOURCE CDATA #REQUIRED>
  <!ELEMENT GRID (CLUSTER | GRID | HOSTS | METRICS)*>
    <!ATTLIST GRID NAME CDATA #REQUIRED>
    <!ATTLIST GRID AUTHORITY CDATA #REQUIRED>
    <!ATTLIST GRID LOCALTIME CDATA #IMPLIED>
  <!ELEMENT CLUSTER (HOST | HOSTS | METRICS)*>
    <!ATTLIST CLUSTER NAME CDATA #REQUIRED>
    <!ATTLIST CLUSTER OWNER CDATA #IMPLIED>
    <!ATTLIST CLUSTER LATLONG CDATA #IMPLIED>
    <!ATTLIST CLUSTER URL CDATA #IMPLIED>
    <!ATTLIST CLUSTER LOCALTIME CDATA #REQUIRED>
  <!ELEMENT HOST (METRIC)*>
    <!ATTLIST HOST NAME CDATA #REQUIRED>
    <!ATTLIST HOST IP CDATA #REQUIRED>
    <!ATTLIST HOST LOCATION CDATA #IMPLIED>
    <!ATTLIST HOST REPORTED CDATA #REQUIRED>
    <!ATTLIST HOST TN CDATA #IMPLIED>
    <!ATTLIST HOST TMAX CDATA #IMPLIED>
    <!ATTLIST HOST DMAX CDATA #IMPLIED>
    <!ATTLIST HOST GMOND_STARTED CDATA #IMPLIED>
  <!ELEMENT METRIC EMPTY>
    <!ATTLIST METRIC NAME CDATA #REQUIRED>
    <!ATTLIST METRIC VAL CDATA #REQUIRED>
    <!ATTLIST METRIC TYPE (string | int8 | uint8 | int16 | uint16 | int32 | uint32 | float | double | timestamp)
#REQUIRED>
    <!ATTLIST METRIC UNITS CDATA #IMPLIED>
    <!ATTLIST METRIC TN CDATA #IMPLIED>
    <!ATTLIST METRIC TMAX CDATA #IMPLIED>
    <!ATTLIST METRIC DMAX CDATA #IMPLIED>
    <!ATTLIST METRIC SLOPE (zero | positive | negative | both | unspecified) #IMPLIED>
    <!ATTLIST METRIC SOURCE (gmond | gmetric) #REQUIRED>
  <!ELEMENT HOSTS EMPTY>
    <!ATTLIST HOSTS UP CDATA #REQUIRED>
    <!ATTLIST HOSTS DOWN CDATA #REQUIRED>
    <!ATTLIST HOSTS SOURCE (gmond | gmetric | gmetad) #REQUIRED>
  <!ELEMENT METRICS EMPTY>
    <!ATTLIST METRICS NAME CDATA #REQUIRED>
    <!ATTLIST METRICS SUM CDATA #REQUIRED>
    <!ATTLIST METRICS NUM CDATA #REQUIRED>
    <!ATTLIST METRICS TYPE (string | int8 | uint8 | int16 | uint16 | int32 | uint32 | float | double | timestamp)
#REQUIRED>
    <!ATTLIST METRICS UNITS CDATA #IMPLIED>
    <!ATTLIST METRICS SLOPE (zero | positive | negative | both | unspecified) #IMPLIED>
    <!ATTLIST METRICS SOURCE (gmond | gmetric) #REQUIRED>
]>
<GANGLIA_XML VERSION="3.0.6" SOURCE="gmond">
<CLUSTER NAME="XTORC" LOCALTIME="1248300236" OWNER="SRT" LATLONG="unspecified"
URL="http://xtorc.csm.ornl.gov">
<HOST NAME="node6.xtorc" IP="192.168.50.6" REPORTED="1248300230" TN="5" TMAX="20" DMAX="0"
LOCATION="XTORC" GMOND_STARTED="1245777829">
<METRIC NAME="disk_total" VAL="39.279" TYPE="double" UNITS="GB" TN="2746" TMAX="1200"
DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_speed" VAL="1995" TYPE="uint32" UNITS="MHz" TN="346" TMAX="1200" DMAX="0"
SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="part_max_used" VAL="27.5" TYPE="float" UNITS="%" TN="46" TMAX="180" DMAX="0"
SLOPE="both" SOURCE="gmond"/>
```

<METRIC NAME="swap_total" VAL="499988" TYPE="uint32" UNITS="KB" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="os_name" VAL="Linux" TYPE="string" UNITS="" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="cpu_user" VAL="0.0" TYPE="float" UNITS="%" TN="45" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_system" VAL="0.0" TYPE="float" UNITS="%" TN="45" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_aidle" VAL="100.0" TYPE="float" UNITS="%" TN="45" TMAX="3800" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_five" VAL="0.00" TYPE="float" UNITS=" " TN="46" TMAX="325" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="proc_run" VAL="0" TYPE="uint32" UNITS=" " TN="156" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="mem_free" VAL="201120" TYPE="uint32" UNITS="KB" TN="65" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="mem_buffers" VAL="114524" TYPE="uint32" UNITS="KB" TN="65" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="swap_free" VAL="499988" TYPE="uint32" UNITS="KB" TN="65" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="bytes_in" VAL="746.50" TYPE="float" UNITS="bytes/sec" TN="205" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="pkts_out" VAL="0.30" TYPE="float" UNITS="packets/sec" TN="205" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_num" VAL="1" TYPE="uint16" UNITS="CPUs" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="disk_free" VAL="36.532" TYPE="double" UNITS="GB" TN="46" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="mem_total" VAL="775116" TYPE="uint32" UNITS="KB" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="cpu_wio" VAL="0.0" TYPE="float" UNITS="%" TN="45" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="boottime" VAL="1235743176" TYPE="uint32" UNITS="s" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="machine_type" VAL="x86" TYPE="string" UNITS="" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="os_release" VAL="2.6.24-16-generic" TYPE="string" UNITS="" TN="346" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="cpu_nice" VAL="0.0" TYPE="float" UNITS="%" TN="45" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_idle" VAL="100.0" TYPE="float" UNITS="%" TN="45" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_one" VAL="0.00" TYPE="float" UNITS=" " TN="46" TMAX="70" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_fifteen" VAL="0.00" TYPE="float" UNITS=" " TN="46" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="proc_total" VAL="43" TYPE="uint32" UNITS=" " TN="156" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="mem_shared" VAL="0" TYPE="uint32" UNITS="KB" TN="65" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="mem_cached" VAL="398916" TYPE="uint32" UNITS="KB" TN="65" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="gexec" VAL="OFF" TYPE="string" UNITS="" TN="46" TMAX="300" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="bytes_out" VAL="15.10" TYPE="float" UNITS="bytes/sec" TN="205" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="pkts_in" VAL="12.43" TYPE="float" UNITS="packets/sec" TN="205" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond"/>
</HOST>

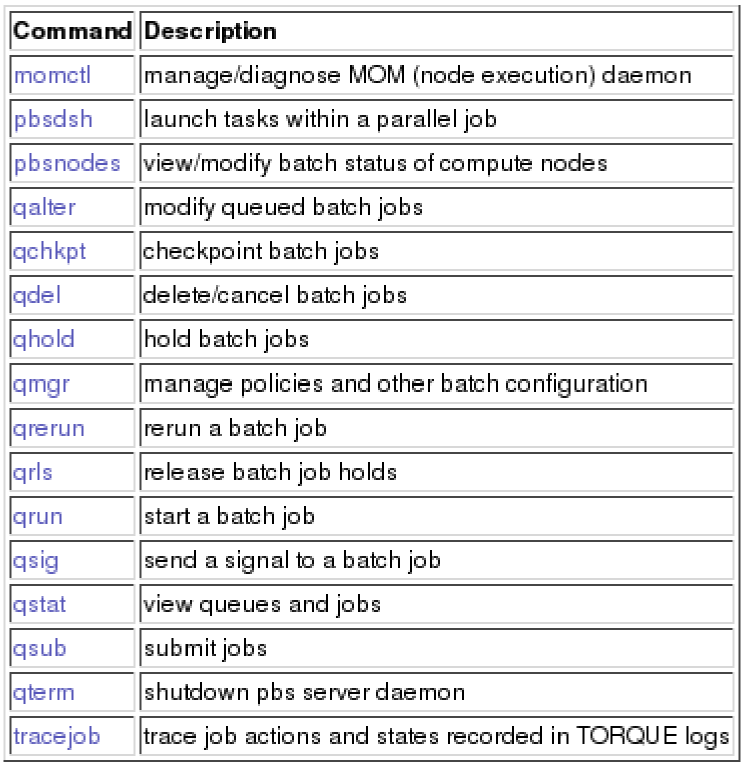
</CLUSTER>
</GANGLIA_XML>

8.8 Linux Log Files Example

```
Sep 21 19:24:47 ubuntu syslog-ng[4908]: syslog-ng starting up; version='2.0.9'
Sep 21 19:24:47 ubuntu kernel: [    0.000000] Initializing cgroup subsys cpuset
Sep 21 19:24:47 ubuntu kernel: [    0.000000] Initializing cgroup subsys cpu
Sep 21 19:24:47 ubuntu kernel: [    0.000000] Linux version 2.6.27-14-generic (buildd@vernadsky)
(gcc version 4.3.2 (Ubuntu 4.3.2-1ubuntu11) ) #1 SMP Tue Aug 18 16:25:45 UTC 2009 (Ubuntu
2.6.27-14.39-generic)
Sep 21 19:24:47 ubuntu kernel: [    0.000000] BIOS-provided physical RAM map:
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 0000000000000000 -
000000000009fc00 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 000000000009fc00 -
00000000000a0000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000000ef000 -
0000000000100000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 0000000000100000 -
00000000b7920000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000b7920000 -
00000000b7930000 (ACPI NVS)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000b7930000 -
00000000b793b000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000b793b000 -
00000000b793d000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000b793d000 -
00000000bb6e7000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bb6e7000 -
00000000bb8e7000 (ACPI NVS)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bb8e7000 -
00000000bbb93000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbb93000 -
00000000bbb9b000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbb9b000 -
00000000bbbbf000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbbbf000 -
00000000bbbcf000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbbcf000 -
00000000bbccf000 (ACPI NVS)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbccf000 - 00000000bbcff000
(ACPI data)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbcff000 -
00000000bbd00000 (usable)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000bbd00000 -
00000000c0000000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000e0000000 -
00000000f0000000 (reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000fec00000 - 00000000fec01000
(reserved)
Sep 21 19:24:47 ubuntu kernel: [    0.000000]  BIOS-e820: 00000000fed10000 -
00000000fed14000 (reserved)
```

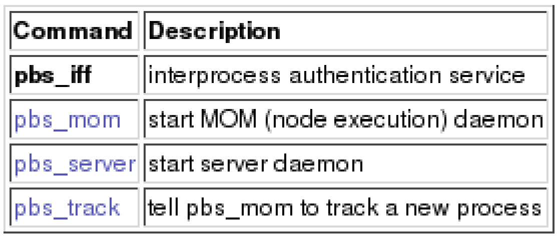
8.9 Torque Commands [34]

Client Commands



Command	Description
momctl	manage/diagnose MOM (node execution) daemon
pbsdsh	launch tasks within a parallel job
pbsnodes	view/modify batch status of compute nodes
qalter	modify queued batch jobs
qchkpt	checkpoint batch jobs
qdel	delete/cancel batch jobs
qhold	hold batch jobs
qmgr	manage policies and other batch configuration
qrerun	rerun a batch job
qrls	release batch job holds
qrun	start a batch job
qsig	send a signal to a batch job
qstat	view queues and jobs
qsub	submit jobs
qterm	shutdown pbs server daemon
tracejob	trace job actions and states recorded in TORQUE logs

Server Commands



Command	Description
pbs_iff	interprocess authentication service
pbs_mom	start MOM (node execution) daemon
pbs_server	start server daemon
pbs_track	tell pbs_mom to track a new process