

# Fault tolerance-enabled HPC resource management with HA-OSCAR framework \*

*Kshitij Limaye, Anand Tikotekar, Box Leangsuksun  
Louisiana Tech University, Ruston, LA 71270, USA*

[{ksl007, aat007, box}@latech.edu](mailto:{ksl007, aat007, box}@latech.edu)

## ABSTRACT

The computing power offered by large-scale clusters is helping address various challenges posed for the research community. In order to reduce the computational time necessary to meet these challenges, the system size is increasing. It is critical to guarantee the full utilization of these resources to meet the challenges and to justify the investment. In Beowulf cluster systems, a service node failure may render the whole system outage. This failure results in loss of the critical services, such as the resource manager service and the user submitted jobs. Unavailability of critical services calls for proactively handling failures at a job-site level, ensuring the system high availability with no loss of user submitted jobs. Only a few services have built-in fault tolerance by saving their own state and having active/backup approach. Therefore, we need to improve fault tolerance for services that lack these features. We aim towards adding fault tolerance to such services by using the underlying HA-OSCAR framework with minimal changes to the services to guarantee their high availability with no loss of state.

## 1. Introduction

In recent years, large-scale clusters have been used to solve computationally intensive problems, such as particle physics and bio computing, in high performance computing (HPC) environments. This large investment needs to be guaranteed against failures, especially failures of the head node, to obtain maximum computational benefit from the compute nodes. Since the failures of a single node could result in a system outage, it is essential to effectively deal with faulty situations in the HPC environment.

To effectively use the computational resources available in large cluster environments,

most clusters have a resource manager/scheduler service to manage and allocate resources. This service manages the user submitted jobs and the resource pool to complete the user requests. The loss of this service would lead to loss of user submitted jobs and hence the computational time spent on them leading to degraded use of resources. The resource manager service needs to provide fault tolerance for a worker node failure, a subcomponent failure, or the failure of the node on which it ran.

The fault tolerance for worker node failures can be achieved with checkpoint/ restart mechanism for jobs running on that resource. A mix of serial and parallel jobs runs on large-scale clusters; checkpoint/restart scheme should be provided for both types of jobs. A common mechanism for resource managers to achieve fault tolerance against service node failures is by check pointing their own state and having a backup scheduler up to date with it. This would imply that the job queue is preserved with no/minimal loss of running jobs when the backup resource manager takes over.

Some job management systems (JMS) employ a run-time library based checkpointing while a very few provide kernel level system checkpointing facility. Most JMSs do not provide all of the above fault tolerance mechanisms; therefore, they are vulnerable to failures. We aim to add fault tolerance to such services through our active/standby HA framework in turn needing minimal changes to the services. While commercial job schedulers/resource managers with active/backup framework and checkpoint support for serial and parallel jobs exist, there is dearth of such solutions in the Open Source community.

High Availability (HA) computing strives to avoid the problems of unexpected failures through active redundancy and preemptive measures. As the head node machine, in Beowulf based clusters, has critical services, such as job schedulers/resource managers, its availability is critical and should be guaranteed.

\* supported by Office of Science, US Department of Energy, *fastOS* program, Grant # DE-FG02-04ER4614.

High Availability Open Source Cluster Application Resource (HA-OSCAR) is a solution that intends to provide high availability and serviceability for the large-scale HPC Linux cluster environment [2]. The failover mechanism in HA-OSCAR proves to be graceful when stateless services are involved; however it is inadequate when stateful services that lack built in fault tolerance, for example job management, are involved [3].

These services have a state. For example in case of resource managers, the state would correspond to the status of each job in the job queue, which needs to be preserved in case of failures to avoid loss of the computation previously completed. This paper details the use of a HA based framework to enhance fault tolerance to a resource management system. Our approach aims to proactively handle failures by replicating the job queue and avoiding loss of computational time spent on jobs by preserving their state till the last checkpoint.

The Torque (Tera-scale Open-source Resource and QUEue manager) resource manager [7], which comes with the OSCAR 4.x [5] cluster bundle, was used as service in question to add fault tolerance by incorporating checkpoint support for parallel jobs and preserving the job queue state in the event of a head node failure. Section 2 describes the related work done in this field. In section 3, we discuss the proposed framework for the fault tolerant resource manager and section 4 describes the conducted experiment and some preliminary results obtained. We conclude in section 5.

## 2. Related Research

LinuxHA [4] is a tool for building HA clusters using data replication as the primary technology. However, LinuxHA only provides a heartbeat and failover mechanism for a flat-structure cluster which does not easily support the Beowulf architecture commonly used by most job sites.

OSCAR is a software stack for deploying and managing Beowulf clusters. Unfortunately, a detrimental factor of Beowulf architecture is the possibility of single-point-of-failure. The cluster can go down completely with the failure of head node. Therefore, there is a need to focus on the high availability aspect of the cluster design.

The recently released HA-OSCAR software stack is one such effort making inroads here. HA-OSCAR deals with availability and

fault issues at the master node with multi-head failover architecture and service level fault tolerance mechanisms for stateless services. In addition, HA-OSCAR provides a flexible and extensible interface for customizable fault management, failover operation, and alert management.

Typically, job management systems consist of two parts, resource managers and dedicated job schedulers. The resource managers normally handle the job submission and management of resources. The resource manager provides the schedulers the user job requirements and a list of available resources. Along with these inputs, the scheduler employs certain scheduling and usage policies to allocate resources for jobs based on various criteria. OpenPBS [6], Torque, SLURM [8], and Condor were some of the resource managers that were studied.

Torque is based on the OpenPBS project, which is no longer supported. Torque is open source software and has added scalability, node fault tolerance, and better logging facilities compared to OpenPBS. The Torque server is still a single point of failure and does not have a standby server monitoring it and taking over after detection of failure. Also, Torque provides no support for checkpointing unless the underlying OS provides it leading to loss of jobs on failure.

Lawrence Livermore National Laboratory SLURM [8] was developed with scalability in mind coupled with fault tolerance and simplistic management. SLURM has an Active/Standby server configuration for fault tolerance. In an event of the outage of the primary SLURM control daemon, the backup controller assumes control with no job loss. The SLURM controller daemon writes its current state to disk when the backup controller takes over to preserve the job queue. Currently, SLURM has no support for checkpointing.

Condor is used for high throughput computing. Condor supports both serial and parallel jobs but provides checkpointing and process migration for serial jobs. The condor central manager is also a single point of failure and a HA solution [10] is being developed to ensure its availability. The failure of Condor Central Manager (CM) leads to an inability to match new jobs and respond to queries regarding job status and usage statistics. Condor attempts to eliminate the single point of failure (i.e. the Condor CM) by having multiple CMs and a high

availability daemon (HAD) which monitors them to ensure that one of CMs is active at all times.

An important scheme to achieve fault tolerance, for running jobs in cluster systems, is checkpoint/restart technique. There are numerous Linux-based checkpoint/restart packages such as BLCR [11], CoCheck [12] and Libckpt [13]. Berkeley Lab’s Linux Checkpoint/Restart project (BLCR) is a kernel-level checkpoint/restart package for multi-threaded applications on a Linux platform. BLCR can be used either as a standalone system for check-pointing applications on a single machine, or as a component by a scheduling system or parallel communication library for check-pointing and restoring parallel jobs running on multiple machines. LAM/MPI [14] based parallel jobs can be check-pointed using BLCR.

### 3. Proposed Framework

As described earlier, a failure at the head node of a Beowulf cluster causes loss of critical services, and can lead to significant downtime and loss of computation. Limaye and Leangsuksun in [15] introduced fault tolerance to the Torque resource manager by preserving the job queue on the backup head node through event-based replication of jobs.

This approach provided fault tolerance by preserving the state of the JMS in spite of a head node failure but lacked fault tolerance for running jobs. Their work required restarting the lastly “running” jobs from scratch leading to loss of computational time (could be order of days) spent on those jobs. This necessitated the provision of fault tolerance for running jobs through a checkpoint/restart mechanism in resource managers, so that we only stand to lose the computation occurring after the last checkpoint.

In this paper, we investigated checkpoint mechanisms for parallel jobs and feasibility study of resource managers to provide per job checkpoint support.

Figure 1 describes the components necessary to make a resource management system fault tolerant. It mainly consists of the *event monitor*, *job updater*, *checkpoint controller*, an event notifying scheduler wrapper, and the HA-OSCAR *monitoring core*. The event generating wrapper notifies the *event monitor* whenever a job is added or completed. The HA-OSCAR monitoring core daemon notifies the *event monitor* on system events and leaves it upon the event monitor to act on them. The *job*

*updater* replicates a job’s information and files to the backup whenever it is added and similarly deletes them whenever it completes, leading to preservation of job queue on the backup. The *checkpoint controller* whose lifetime is equal to the job runtime, checkpoints the job periodically as well as when a critical system event as notified by the *event monitor*.

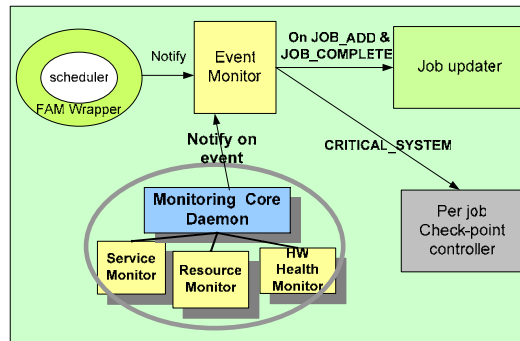


Figure 1: Checkpoint support System Components

#### 3.1. HA-OSCAR Monitoring Core

Figure 1 illustrates the HA-OSCAR smart failover mechanism with checkpoint support. The framework consists of 3 components: the *event monitor*, *job monitor* and the *backup updater*. The event monitor, using the HA-OSCAR monitoring core, analyzes critical system events, for example repeated service failure, memory leaks, and system overload. The HA-OSCAR’s service monitoring core determines which services are down and subsequently acts upon them. The hardware monitoring daemon of HA OSCAR receives important and critical events through the IPMI enabled hardware. The event monitor further analyses these events according to the policy set out by the administrator. The policy consists of deciding which events are considered critical and which events necessitate a failover. To exemplify, consider that HA OSCAR’s Hardware monitoring daemon receives event such as “processor temperature exceeded non critical threshold.” This event, although signaling an impending problem, may not necessitate a failover at that moment. If this event is followed by an event of the form “processor temperature exceeded critical threshold,” then a failover might become necessary. As noted earlier, the policy here is configurable. The events received by the Hardware monitor are in the form of alert strings, which are forwarded to the event monitor. A checkpoint signal is placed if the event monitor

decides an event signifies a failover. In this case, all the running jobs will be checkpointed.

### 3.2. Job Queue Preservation

Figure 2 depicts the algorithm used by the Event Monitor. As discussed earlier the event monitor receives event notification from the HA-OSCAR monitoring core and the event generator written over scheduler. The event monitor receives two types of events from the event generator, `JOB_ADD` and `JOB_COMPLETE`. On receiving the `JOB_ADD` and `JOB_COMPLETE` event from the scheduler wrapper, the event monitor triggers the *job updater* to update the job addition/completion to the backup. The event monitor also receives `SYSTEM` events from the HA-OSCAR monitoring core. Based on the frequency and severity of these events received from the HA-OSCAR monitoring core and the predefined policy, the *event monitor* decides whether to alert the *Ckpt controller* with a `CRITICAL_SYSTEM` event.

Figure 3 describes the steps involved in achieving the checkpoint aware scheduler service. When a job is submitted to the queue, two things happen simultaneously; the event generator wrapper notifies the event monitor about the job addition and the scheduler invokes the pre job setup script to perform any initialization work. The event monitor invokes the job updater to update the backup with the newly added job. The job updater replicates the job files with a “held” status. Meanwhile, the per job setup script spawns a checkpoint controller for that job as a separate process and exits.

Most schedulers support a job setup script to run before the job starts and a cleanup script to run after the job finishes. The *Ckpt controller* is created by the pre job setup script and is provided the relevant jobID and the user name of the job submitter. The *Ckpt controller* remains in existence until the job is executed. The *Ckpt controller* is responsible for periodically check-pointing the job and whenever notification of a `CRITICAL_SYSTEM` event is fired by the event monitor. Additionally, the *Ckpt controller* creates a job specification for the check-pointed job and transfers it to the backup. This specification file is similar to that submitted for the original job, differing only in the execution command to be run. The execution command is replaced by the checkpoint restart command in order to start the job from the last check-point.

```

Update_Backup_with_ckpt (event_type, jobID)
{
  If (event_type == 'JOB_ADD')
  {
    Add the new job files to temp dir
    If (job submitted through Grid)
      Map scheduler jobID to Globus assigned jobID
  }
  If (event_type == 'JOB_COMPLETE')
  {
    Remove completed job's files from temp dir
  }
  if (event_type == 'SYSTEM')
  {
    record and store the alert that generated the event
    filter recently received events for criticality
    if (combination of events suggest a Failover)
    {
      notify chkt controllers for all running jobs to
      checkpoint
      Cause a SYSTEM_DOWN to start the
      failover process
    }
  }
}

```

Figure 2: Event Monitor Algorithm

After the pre job setup script has finished execution, the job starts running. The Ckpt controller spawns another thread to listen for notification of `CRITICAL_SYSTEM` events from the event monitor. It also periodically check-points the job based on the pre-defined checkpoint interval. Whenever it first creates a checkpoint, it also creates a specification for the check-pointed job and transfers it to the backup.

When the job completes, the job cleanup script is invoked, which removes the check-point files and the specification file created for the job since they are unnecessary due to normal job completion. Whenever a job completes, the scheduler wrapper notifies the event monitor about the completion. The event monitor invokes the job updater to remove the temporary job files related to the completed job from the backup. These steps cause the removal of the original job files as well as the checkpoint files from the backup completing the job cleanup process on successful job completion.

The above system assures that for every job a checkpoint controller is created and which would checkpoint the job periodically and on critical system events, thus minimizing the computational time lost due to a failure.

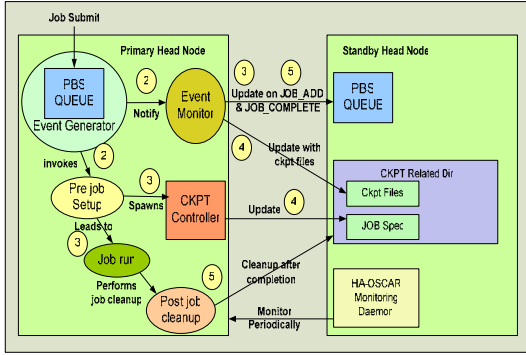


Figure 3: Non Failure Scenario

### 3.3. Recovery from Failure

The HA-OSCAR backup node monitors the primary periodically and takes over in case of an outage. In the event of an outage, the backup assumes control and invokes a `Serverdown.alert` script in order to recover from the head node failure. Figure 4 explains the steps taken to preserve the job queue order and start the checkpointed jobs.

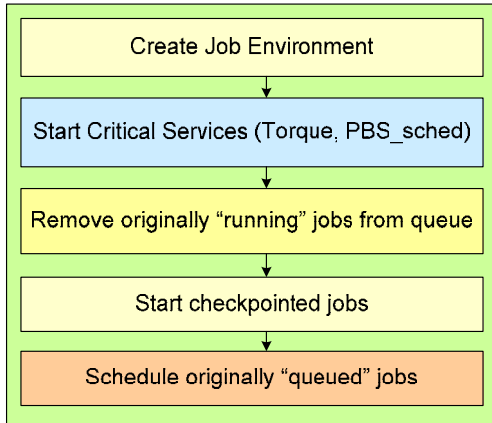


Figure 4: Steps Performed by Standby Server to Recover

Certain scheduling systems, like Torque, create a per job based environment before the job starts running. To achieve transparent checkpointed and restarted job, an environment identical to the failed primary head node must be recreated on the backup after the failover. The recreation of the environment involves the creation of LAM daemons with suffixes containing the jobIDs of the last running jobs. This facilitates the execution of checkpointed jobs.

The critical services, Torque and PBS\_sched, are started after the job environment has been created. The *job updater* replicates a job's temporary job files to the backup while the

*ckpt controller* updates the backup with the check-point files and the new job specification file for that job. Now that the new job specification (to restart the job from last checkpoint) exists on the backup, we need to remove the original job specification to avoid restarting of job from scratch. If this is not done, then the same job would be restarted from scratch as well as its check-pointed version will be in queued state.

After the older job specification files have been removed, the checkpointed job specifications are used to restart the jobs. Since, the jobs are replicated to the backup by the *job updater* with status "held.", the execution is at the discretion of the backup. After the checkpointed jobs have been restarted, the rest of the "held" queue is released and those jobs are queued for execution.

This preserves the job queue sequence to what it was when failure occurred. As discussed, the backup stays updated with the jobs on the primary due to the per-job replication and checkpoint of "running" jobs. The last "running" jobs will resume from their last checkpoint on the standby sever.

## 4. Results and Analysis

Figure 5 shows a comparison of various recovery approaches used in clusters when faced with a system outage. In the case of recovery scenario using HA-OSCAR, the standby node assumes control over the failed primary and starts lastly running jobs from their last checkpoint followed by rest of jobs in queue, preserving the sequence. The critical event based checkpoint approach causes the checkpoint to be placed before the system goes down. So the amount of computational time lost for a job is drastically reduced. This idea is conveyed in figure 5, where the total time for a job to complete after the head node crashes would only be  $T_R + R'_T$ .

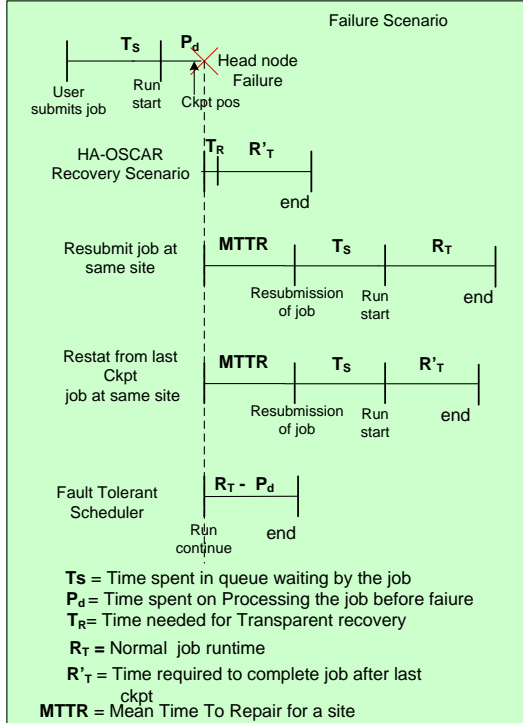


Figure 5: Recovery Analysis

Another recovery approach would consist of resubmitting the lost jobs to the cluster. In this approach, the lost job (running when system failure occurred) would have to face the queue delay again and would have to wait for the system to recover. This leads to a delay of  $MTTR + T_s$  + computational time lost due to no checkpoint. In a similar approach with checkpoint support the delay would still consist of  $MTTR + T_s$  which would be substantial. The delay for a job to restart on the backup node in our solution would be only  $T_R$  as compared to  $(MTTR + T_s)$  in the case where checkpointing is enabled. With  $MTTR \gg T_R$ , this is a clear advantage over above recovery approaches.

Our goal is to improve fault tolerance to a resource management using a HA framework with minimal changes to the service. In our approach, not only have we addressed fault tolerance for the head node, but also the compute nodes as they cause parallel jobs to stall. Most resource managers do not handle the worker node failures. In fact, our approach requires minimal changes to the service and no change to users' application.

## 5. Conclusion

As the system size increases in order to meet new computational challenges, it becomes

imperative to guarantee their continuous utilization. In this paper, we discussed the application of a HA-OSCAR based HA framework to enhance fault tolerance for the resource manager service. Our approach includes the preservation of the job queue and its execution sequence when failover occurs. Furthermore, with the addition of checkpoint/restart facility for serial and parallel jobs in the resource manager, the computational outage is reduced to the least. The periodic checkpoint approach complimented by the event based checkpoint helps guarantee the preservation of the state of every running job until the impending failure occurs. The per job based checkpoint approach coupled with the HA framework guarantees minimal loss of computational time. Our solution ensured fault tolerance with minimal changes to the service. The results show that the coupled effect of the HA framework and fault tolerant service would lead to a better aggregate system performance in a true sense.

## 6. References

- [1] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny, "Condor - A Distributed Job Scheduler", *Beowulf Cluster Computing with Linux*, The MIT Press, 2002. ISBN: 0-262-69274-0.
- [2] C. Leangsuksun *et al*, "A Failure Predictive and Policy-Based High Availability Strategy for Linux High Performance Computing Cluster", The 5th LCI International Conference on Linux Clusters, 2004.
- [3] K. Limaye, C. B. Leangsuksun, *et. al*, "Job-Site Level Fault Tolerance for Cluster and Grid environments", *the 2005 IEEE Cluster Computing*, Boston, MA, September 27-30, 2005.
- [4] LinuxHA Clustering Project, <http://www.linuxha.net//index.pl>
- [5] John Mugler, *et.al*. "OSCAR Clusters", Proceedings of the Ottawa Linux Symposium (OLS'03), Ottawa, Canada, July # 23-26, 2003.
- [6] Ibeaus Bayucan, Robert L. Henderson, *et al*, "Portable Batch System External Reference Specification", MRJ Technology Solutions, May 1999.
- [7] The Torque Resource Manager, <http://www.clusterresources.com/products/torque/>
- [8] SLURM: Simple Linux Utility for Resource Management,

- [http://www.llnl.gov/linux/slurm/slurm\\_design.pdf](http://www.llnl.gov/linux/slurm/slurm_design.pdf)
- [9] Dobriša Dobrenić *et al.*, “Job Management System Analysis”, 6<sup>th</sup> CARNet User Conference, Sept 24-27 2004 Zagreb, HR.
  - [10] Adding high availability to Condor Central manager,  
[http://dsl.cs.technion.ac.il/projects/gozal/project\\_pages/ha/ha.html](http://dsl.cs.technion.ac.il/projects/gozal/project_pages/ha/ha.html)
  - [11] K.M. Chandy, “A Survey of Analytic Models for Rollback and Recovery Strategies,” *Computer*, vol. 8, no. 5, pp. 40-47, 1975.
  - [12] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In Proceedings of the 10th International Parallel Processing Symposium (IPPS '96), Honolulu, Hawaii, 1996.
  - [13] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent Checkpointing under Unix. In Usenix Winter 1995 Technical Conference, page 213-224, 1995.
  - [14] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. “The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing.” In LACSI Symposium, October 2003.
  - [15] “Reliability-aware Resource Management for Computational Grid/Cluster Environments” K. Limaye, C. Leangsuksun *et al.*, In the 6th IEEE/ACM International Workshop on Grid Computing, Seattle, WA, USA, November 2005.