



THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-11-01

**Adaptive Resource Remapping In
Virtualized Environments -
Framework**

Muhammad Atif, Peter Strazdins

May 2011

ANU Computer Science Technical Report Series

This technical report series is published by the Research School of Computer Science, College of Engineering and Computer Science, The Australian National University. Previously published by the School of Computer Science, and prior to 2009 this series was published as the Joint Computer Science Technical Report Series by the Department of Computer Science, FEIT, and the Computer Sciences Laboratory, RSISE, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Research School of Computer Science
College of Engineering and Computer Science
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical-DOT-Reports-AT-cs-DOT-anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

<http://cs.anu.edu.au/techreports/>

Recent reports in this series:

- TR-CS-10-01 Duncan Stevenson, Garry Warne, Mai Eames, Maurice Hennessy, Dan Penny, Jim Wilkinson, and Simon Pase. *Delivery of Postgraduate Medical Training to Developing Countries - A Pilot Study*. September 2010.
- TR-CS-09-02 Daniel Frampton, Stephen M Blackburn, Luke N Quinane, and John N Zigman. *Efficient Concurrent Mark-Sweep Cycle Collection*. October 2009.
- TR-CS-09-01 Peter Christen, Ross Gayler, and David Hawking. *Similarity-Aware Indexing for Real-Time Entity Resolution*. August 2009.
- TR-CS-08-03 Jie Cai, Alistair P. Rendell, Peter E. Strazdins, and H'sien Jin Wong. *Predicting Performance of Intel Cluster OpenMP with Code Analysis Method*. November 2008.
- TR-CS-08-02 Paul Thomas. *Implementation of PIS*. June 2008.

Adaptive Resource Remapping In Virtualized Environments - Framework

Muhammad Atif, Peter Strazdins
Dept. of Computer Science
The Australian National University
Canberra, 0200, Australia

1 Abstract

Cluster computing is going through an evolution from single processor systems to multi core SMP systems. This has resulted in lower cost/performance ratio for the commodity of the shelf clusters (COTS). Compute farms comprising of clusters of commodity-off-the-shelf processors tend to become heterogeneous over time. Different CPU architectures, memory capacities, communication and I/O interfaces of the participating compute nodes often result in under or over utilization of the compute resources.

This trend has coincided with the resurgence of virtualization technology. The virtualization technology is receiving widespread adoption mainly due to the potential benefits of server consolidation and isolation, flexibility, security and fault tolerance. Virtualization also offers other benefits, which include development/testing of applications, live migration and load balancing. Virtualization has also generated considerable interest in High Performance Computing (HPC) community dealing with COTS. This is mainly due to the reasons of high availability, fault tolerance, cluster partitioning and balancing out conflicting user requirements.

We believe that one can leverage the virtualization environment to achieve reduced job turn around times, especially in the case of COTS. These clusters are highly heterogeneous in nature which is due to the presence of different CPU architectures, available memory and the communication interfaces. Different CPU architectures, memory capacities, communication and I/O interfaces of the participating compute nodes present many challenges to the job schedulers and often result in under or over utilization of the compute resources. For this, the application programmers have to specifically write the application for the set of compute nodes, which is time consuming and is not a scalable option.

In this research, we have investigated resource scheduling in the compute clusters with the perspective of dynamic resource remapping. Our approach is to profile each job in the compute farm at runtime, and arrive at a near optimal

resource map for each job. We then migrate the jobs to the best suited compute nodes to improve the overall through put of the compute farm. For this, we have developed a novel heterogeneity and virtualization aware profiling framework, which is able to predict the CPU and communication characteristics of the high performance scientific applications.

2 Introduction

Compute farms, whether for research department clusters, enterprise grids, data centers or supercomputing facilities, tend to become heterogeneous over time. This is due to incremental extension over a period of time and/or particular nodes being purchased for users with particular needs. This heterogeneity is not surprising given the wide range of different processor, memory and network technologies that become available [12, 16] and the relatively small price difference between these various options. After couple of upgrade cycles, the compute farm becomes a heterogeneous compute farm (HC) constituting of a federation of homogeneous sub-clusters.

Parallel applications (scientific or otherwise) have varied computation and communication requirements depending on the domain and the nature of their algorithms. For instance some, applications are floating point intensive while others can be memory or communication intensive. This diverse nature of applications results in varied execution time in the compute farm due to heterogeneity of the nodes. On a heterogeneous cluster where only half of the nodes are linked via an expensive high performance network, it is possible for a parallel application that does little inter-node communication to end up running on the faster part of the cluster, while another application that would greatly benefit from a faster network is left to run on a slower half of the cluster.

Application programming also poses a significant challenge to programmers in heterogeneous clusters compared to the homogeneous systems. A good parallel application for a homogeneous distributed memory multiprocessor system tries to evenly distribute computations over the available processors [12]. If processors run at different speeds, the faster processors will complete their part of computation and begin to wait for the slower processors at points of synchronization and data transfer, a case we call ‘under utilization’ of the compute node. Therefore, the total time of computations will be determined by the time elapsed on the slowest processor [12].

The issue of effective mapping (scheduling) of parallel applications onto such heterogeneous systems is therefore of great interest to researchers. The problem is NP complete [14] and several research studies have addressed this problem by developing heuristic techniques [14, 6, 11]. These heuristics require the scheduler to be aware of the application characteristics and are static in nature. This in turn requires the application programmer to profile and analyze the application prior to job submission. The static nature of this approach prevents dynamic

load balancing within the compute farm.

Another approach to address the computation and communication imbalance of the nodes in an HC is to adapt the parallel application according to the heterogeneity of the compute farm. Here, the parallel application is required to distribute computations unevenly to account for the varied speed and architecture of processors [12, 16]. The load balancing of such systems require a considerable effort from the programmers perspective [20] and the solutions are not generic in nature. In the case of workload, changes in the source code changes are required, which is difficult and time consuming. In order to load balance an application, the programmer is required to determine the application characteristics and performance modeling is required.

In both cases, the allocation of nodes to a parallel job in an HC requires some sort of performance modeling techniques. In performance modeling, an application is profiled to gain an understanding of its performance characteristics. These characteristics are then encapsulated into a set of formulas [9]. The performance models are then evaluated on the different compute nodes and sub-networks to determine the expected speedups (or slowdowns) on various hardware architectures/nodes.

Fine grained performance modeling is capable of reasonably accurate prediction but the associated cost of profiling can be very high in terms of the wall-clock time of the job [16, 20]. Due to these costs, these techniques must be applied to applications in an ‘offline’ mode. The application or some of its iterations are profiled on a set of hardware and scheduling decisions are based on the resultant profile metrics. In the case of a workload change, the application behavior changes and the application needs to be profiled again.

In this report, we are presenting the design, implementation and initial results of our resource remapping framework, which is able to exploit the heterogeneity in a compute farm to improve throughput. We deal with this issue of heterogeneity by breaking the heterogeneous compute cluster into a number of homogeneous sub-clusters. The runtime characteristics of the applications are determined with the combination of hardware performance counters/units (PMUs) and the profiling interface to MPI (PMPI). This enables us to predict the performance of the running MPI applications on all the other sub-clusters present in our heterogeneous compute farm. All this is done without the need of changing the application binary or requiring off-line profiling and analysis. We then propose the best suited sub-cluster for the compute job on the compute farm and migrate the job to the proposed sub-cluster to improve the overall throughput and the average waiting time.

3 Scope

In this technical report, we discuss the mathematical model, implementation and experimental results of our framework called ARRIVE-F (Adaptive Resource

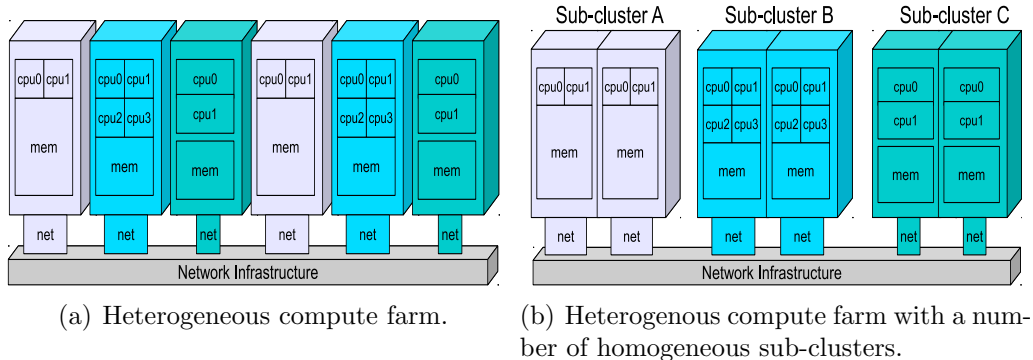


Figure 1: Pictorial view of heterogeneous compute farm.

Relocation in Virtualized Environments).

4 Approach

As discussed, the HC tends to become heterogeneous with each hardware upgrade. To deal with the heterogeneity in such compute farms, we divide the HC into a number of sub-clusters based on the CPU architecture and frequency, available memory and the communication network’s bandwidth, as shown in Figure 1.

The compute jobs are dispatched cluster-wise to the compute farm. No job is dispatched among the two sub-clusters. This ensures that jobs do not run into the load-imbalance introduced by heterogeneity of the compute nodes. Each active job is profiled in the compute farm. Using our model, presented in Section 5, we are then able to propose the best suited sub-cluster for the job. We then migrate the jobs in a way that improves the throughput of the compute farm.

To enable migration, the HC is virtualized using Xen hypervisor. The physical machines are virtualized such that each CPU is allocated a single VM. This ensures that the minimum grain for the migration is a process.

5 Framework

The objective of this framework is to increase the overall throughput of the computer cluster by improving the turn-around times of selected jobs that are dispatched to an *undesired* sub-cluster. The live migration facility provided by Xen is utilized to move the compute job to the best suitable hardware.

The total time T taken by a parallel job P with a total of N processes can be described as a function of CPU, memory, communication and I/O subsystems of each process belonging to job.

$$T(P) = f(CPU(0..N-1), Mem(0..N-1), Com(0..N-1), IO(0..N-1)) \quad (1)$$

Based upon equation 1, we have divided the framework into four sub-models,

The computational model (CPU) , the memory model, the communication model and the I/O model.

The ‘I/O model’ determines the performance of the I/O subsystem minus the communication I/O . These measurements include hard disk I/O and are beyond the scope of this research. The other models are presented in the following sections.

5.1 Assumptions

Our scheduling and profiling framework is targeted towards iterative scientific applications with runtime in the order of minutes or larger. We need applications to run for such periods of time in order to recover the time lost during migration. As the most of the scientific applications are iterative in nature [20] and typically run for hours, these assumptions are realistic in most real world scenarios. As described above, we do not cater for the disk I/O intensive jobs.

We have divided our compute farm into a number of sub-clusters on the basis of hardware specifications. E.g. Cluster ‘A’ represents a 4×2 cluster of AMD Phenom II machines with Gigabit Ethernet interfaces and Cluster ‘B’ represents a 4×4 Opteron 270 cluster. No job is dispatched among the two different hardware platforms or compute clusters. Each virtual machine (VM) is provided with exactly one CPU, i.e. a four CPU machine will have four VMs. One VM per CPU requirement enables us to keep the minimum grain of migration to one process.

We also assume that the approximate wall clock time is provided at the time of submission. It should be noted that the approximate wallclock time is a basic requirement of the widely used backfill algorithm. In section 5.6, we discuss how this assumption can be relaxed.

5.2 Computational Model

The computation model is responsible for generating the CPU profile of the application. We profile all the processes on the compute cluster for the characteristics which are exposed through the hardware performance counters. The profiles include (but are not limited to) the floating point operations rates and the L1/L2 cache miss rates. Our main focus is to identify either the possible hardware limitations which are preventing an optimal application run, or the applications which are ‘under’ utilizing the hardware resources.

To obtain the computation time for a process belonging to a parallel application ‘ j ’, executing on a distinct cluster ‘A’, the job is profiled for a specific time period of ‘ τ ’ seconds. The performance counter events are weighted by their time penalties and added to give the total time. The value of τ can be any positive value large enough to cover at least a single iteration of the parallel application. Note that we take an average of the hardware performance counter events for all the processes, as there is no significant variation in these events between processes

for the scientific applications that we have studied. Once the hardware performance counter (Pctr) events are obtained, we substitute them in the Equation 2 to obtain the CPU time for each process.

$$t_{A,j}^P = \sum_i Pctr_{A,j,i} \times \frac{Cycles_{A,i}}{f_A} \quad (2)$$

where $Pctr_{A,j,i}$ is the count of a specific performance counter event ‘ i ’ (e.g. L2 cache misses or floating point operations) performed by job j . $Cycles_{A,i}$ is the total number of the CPU cycles required to perform the task identified by the $Pctr_{A,i,j}$. E.g. in the case of floating point operations $Cycles_{A,i}$ represents the average number of cycles required to perform a single floating point operation. f_A denotes the CPU frequency of sub-cluster ‘A’. The fraction $\frac{Cycles_{A,i}}{f_A}$ is a constant for the sub-cluster ‘A’.

In order to predict the computation time for the job j on a different cluster ‘B’, we simply substitute the hardware dependent fraction in Equation 2.

$$\tilde{t}_{B,j}^P = \sum_i Pctr_{A,j,i} \times \frac{Cycles_{B,i,j}}{f_B} \quad (3)$$

Note that this time is calculated with respect to the time τ , and that we assume that the event counts will remain approximately the same on cluster ‘B’ (i.e. $Pctr_{B,j,i} \approx Pctr_{A,j,i}$). This may not necessarily hold for events such as cache misses, and may result in some inaccuracy into the prediction.

5.3 Communication Characterization of Parallel Programs

To determine the communication characteristics of a process belonging to a parallel job, we use the MPI profile wrappers known as PMPI. PMPI is an MPI standard and is present in most of the MPI-2 compliant implementations [1].

The total time spent by a process due to communication $t_{A,j}^C$ for a given time period τ is given by:

$$t_{A,j}^C = t_{A,j}^B + t_{A,j}^N \quad (4)$$

where the superscripts B, N and C represent blocking, non-blocking and total (blocking + non-blocking) communication respectively. $t_{A,j}^B$ is the time process j has to wait for the blocking sends or receives to complete. This time is directly related to the bandwidth and latency of the underlying network infrastructure. $t_{A,j}^N$ is the time a process ‘waits’ for the non-blocking communication to finish. A non-blocking send or receive is typically followed by computation and then a wait. Therefore, the time a process has to wait for the non-blocking communication to finish not only depends on the communication bandwidth and latency of the underlying network but also the computational power of the CPU.

In order to determine the time spent by the process in blocking communication, we log the frequency of distinct messages according to the message size. Let ‘ $n_j^B(s)$ ’ be the total number of distinct messages of size ‘ s ’ in a time period ‘ τ ’.

The total time spent by the process j executing on sub-cluster ‘A’ in blocking communications, $t_{A,j}^B$, is given by:

$$t_{A,j}^B = \sum_s n_j^B(s) \times l_A(s) \quad (5)$$

where $l_A(s)$ is the communication network’s latency at size ‘s’ for the sub cluster ‘A’. The latency is determined through micro-benchmarks. To predict the time spent in blocking communication for the target cluster ‘B’, $l_A(s)$ is replaced with the target cluster’s network latencies in the Equation 5.

$$t_{B,j}^B = \sum_s n_j^B(s) \times l_B(s) \quad (6)$$

The blocking collectives are also modeled in the similar way. For example, MPI_Alltoall is a collective operation where all processes send the same amount of data to each other and receive the same amount of data from each other. Essentially, this collective operation is a combination of MPI_Send and MPI_Recv operations. This means each process sends a blocking message of size s , to all the other processes i.e. message of size ‘s’ is send ‘NP’ times for each process, where NP is the total number of processes in a parallel application. Similarly, for the receive operation, each process receives a message of size ‘s’ from every other process in the communicator. Therefore for each MPI_Alltoall operation, $n_j^B(s)$ is incremented by the total number of processes, once for send and once for receive.

Predicting accurate communication times for non-blocking communication is difficult because of the overlap of communication and computation [20]. A non-blocking communication is usually followed by an MPI_Wait. Here, the time that an application has to wait for the communication to complete is more relevant than the network latency. We have devised a lightweight approximation for the non-blocking communication. This is done by logging each non-blocking message and comparing it against the corresponding MPI_Request.

The time a process waits for all the non-blocking communication to finish during the time period τ can be given as

$$t_{A,j}^N = \sum_s n_j^N(s) \times w_A(s) \quad (7)$$

where $n_j^N(s)$ represents the number of non-blocking messages of distinct size ‘s’. $w_A(s)$ is the average waiting time for all the messages of size ‘s’. $w_A(s)$ is calculated by logging each non-blocking send or receive and calculating the corresponding time a process had to wait for that particular message to complete. In order to determine the waiting time for cluster ‘B’ with a different interconnect, we simply multiply the term $n_j^N \times w_A(s)$ in equation 7 with the ratio of latencies of cluster ‘B’ to cluster ‘A’ as shown in Equation 8.

$$\tilde{t}_{B,j}^N = \sum_s n_j^N(s) \times w_A(s) \times \frac{l_B(s)}{l_A(s)} \quad (8)$$

where $\tilde{t}_{B,j}^N$ represents the approximate time the process waits for all the non-blocking communication to finish during the time period ‘ τ ’.

Xen utilizes page flipping mechanism (bridge architecture) for intra-domain communication [10]. A co-located application, i.e. an application with all the processes on the same VMM does not use the external network cards for its communication. As shown in [7], compared to the shared memory the intra-domain communication is very slow. However, compared to utilizing network cards for intra-domain communication (Separate Bridges), this method is faster. Therefore, the framework does not model the intra-domain communication and treats it as a shared memory communication channel. While making migration predictions 5.5, if the framework identifies an application to be co-located, its communication latency is equated to zero. As we show in Sections 7 and 8, this simple methodology is quite effective. However, if required, this can be modeled like any other network infrastructure.

5.4 Memory Utilization

The swap partition utilization by any process belonging to an HPC application has a time penalty in the order of minutes even for the application with a wall clock time in seconds. Our framework does not predict the performance of such a case; however, it is able to detect this and then take appropriate actions so that the application can avoid thrashing. The implementation details of memory utilization are given in Section 6.

5.5 Predicted Execution Time

The time gained or lost by the job if it was executed on cluster ‘B’ can be obtained by subtracting the predicted computation and communication times for sub-cluster ‘B’ from the profile times of sub-cluster ‘A’.

The predicted time saved or lost can be obtained by subtracting Equation 3 from Equation 2, Equation 6 from Equation 5 and Equation 8 from Equation 7 and adding the results as follows:

$$t_{A \rightarrow B,j} = (t_{A,j}^P - \tilde{t}_{B,j}^P) + (t_{A,j}^B - t_{B,j}^B) + (t_{A,j}^N - \tilde{t}_{B,j}^N) \quad (9)$$

where $t_{A \rightarrow B,j}$ is the predicted time saved or lost by the job j on sub-cluster ‘B’ for every τ seconds. Here a negative value means that application will run slower on the sub-cluster ‘B’.

In order to determine the execution time $T_{A \rightarrow B,j}$ of the job j on sub-cluster ‘B’, $t_{A \rightarrow B,j}$ is multiplied with total number of τ blocks in the actual run of application on sub-cluster ‘A’.

$$T_{A \rightarrow B, j} = t_{A \rightarrow B, j} \times \frac{T_{A, j}^{act}}{\tau} \quad (10)$$

where $T_{A, j}^{act}$ is the actual time taken by job j on sub-cluster ‘A’. Equation 10 gives the expected execution time of job j on sub-cluster ‘B’ based on the profile data calculated on sub-cluster ‘A’. However, note that $T_{A, j}^{act}$ is not possible to compute unless the application completes its execution on cluster ‘A’. This value cannot be utilized by our framework because of our assumption that no prior application information is available. In the coming section, we show how we deal with this issue.

However, we have use the value of $T_{A, j}^{act}$ in Section 7.3 to compare the accuracy of our prediction with the actual benchmark execution time.

5.6 Migration Prediction

As discussed, the main objective of the framework is to increase the throughput of the compute farm by migrating jobs to the best suited sub-clusters. This requires the framework to compute the predicted execution time of each active job on all the sub-clusters. As the sub-clusters in the compute farm might be busy entertaining a number of other parallel jobs (e.g. k, l, m), the framework inspects all the jobs and treat all the sub-clusters as ‘potential targets’. If the potential target cluster is free, then the job is migrated from the source cluster to the target cluster. However, if the potential target cluster is busy servicing another job, the impact of job migration on both the sub-clusters is calculated, i.e. the time lost of by jobs during the migration.

The framework migrates the running instances of jobs; hence the remaining execution time of the job is used rather than the actual execution time. The remaining time can be approximated by taking the difference between the estimated execution time T_j^{est} and the elapsed time T_j^{elap} as shown:

$$T_j^{rem} = T_j^{est} - T_j^{elap} \quad (11)$$

The estimated execution time of the job j for sub-cluster ‘B’ can be obtained by given by replacing $T_{A, j}^{act}$ in Equation 10 by T_j^{rem} as shown:

$$T_{A \rightarrow B, j} = t_{A \rightarrow B, j} \times \frac{T_j^{rem}}{\tau} \quad (12)$$

The aggregated time saved on the compute farm when the jobs j and k are swapped can be calculated from Equation 13 as follows:

$$T_{j, k}^{A \leftrightarrow B} = \eta_j T_j^{A \rightarrow B} + \eta_k T_k^{B \rightarrow A} - T^M \quad (13)$$

where $T_{j, k}^{A \leftrightarrow B}$ gives the total time gained by the compute farm in the case of migration of jobs to the respective sub-clusters. A negative value means that the proposed migration will result in reduced throughput of the compute farm. η_j

and η_k are the total number of processes of the corresponding MPI jobs. The term $T_j^{A \rightarrow B}$ gives the time gained or lost by job j if it was moved from cluster ‘A’ to cluster ‘B’; it is not the representative of the total time saved/lost by the sub-cluster ‘A’. To normalize it with respect to nodes of cluster ‘A’, we must multiply the term with the total number of processes of job j . We deal with the job k in the similar way. T^M gives the average time stretch introduced in the wall clock time of jobs j and k by the migration. The value of T^M depends on the network bandwidth and CPU frequency of the involved sub-clusters as well as the network utilization and store operations of the involved jobs [8]. T^M can be determined through the profile data but, for simplicity, we use a linear approximation $T^M = T^m(\eta_j + \eta_k)$, where T^m is the average wall clock time stretch introduced by a single process migration. For a compute job with a large number of nodes, this approximation may be pessimistic. During the migration the scheduler is locked from dispatching jobs to the involved VMMs. We have a further discussion on this in Section 6.1

Equation 13 is based on the assumption that the estimated execution time provided by the user is correct. However, obtaining an accurate estimated time for a job on the heterogeneous cluster is not easy as the job can run on a different set of hardware each time. Users also tend to over-estimate the application runtime to avoid getting their jobs killed [19]. This will result in over-estimation of the predicted time and thereby skewing the migration decisions in the favor of over-estimated jobs.

To remove the dependence on the users’ runtime estimates, the estimated remaining time of the job obtained from Equation 11 is used as a reference only. This value only enables the framework to determine whether the application has sufficient runtime left to at least cover the wall-clock time stretch introduced by migration. We use a fixed value of time T^β to predict the impact of migration. If the remaining time of the job is expected to be more than T^β , only then it is considered for the migration. The value of T^β is partially based on T^M and T_j^{rem} as shown.

Let T^M represent the time average penalty on the wall clock time of parallel jobs when migrated from one sub-cluster to another. The job j should at least run for $\frac{T^M}{t_{A \rightarrow B, j}}$ profile time blocks (τ) to recover the cost of migration. To post substantial gain, the application should continue to run for β time blocks.

$$T^\beta = \begin{cases} \beta\tau & \frac{T_j^{rem}}{\tau} > \frac{T^M}{t_{A \rightarrow B, j}} \\ 0 & otherwise \end{cases} \quad (14)$$

where β is a configurable parameter. The value of β can be adjusted according the migration overheads and the average job length in the compute cluster such that $\beta\tau > T^M$. The predicted execution time on the destination sub-cluster for the next β blocks can thus be calculated as:

$$T_{A \rightarrow B, j} = t_{A \rightarrow B, j} \times \frac{T^\beta}{\tau} \quad (15)$$

The execution time $T_{B \rightarrow A, k}$ for the job k is computed in the similar way. The predicted times are substituted in Equation 13 to compute the total time saved/lost by the compute farm. We also introduce a threshold value T^{Thresh} , which is the minimum time gain expected to be posted by the compute farm in order for migration sequence to proceed. The value of T^{Thresh} is again a configurable parameter, and is a percentage of $\beta\tau\eta(j, k)$. For example a value of $T^{Thresh} = 0.1\beta\tau\eta(j, k)$ suggests that the jobs j and k should only be migrated if the expected gain in time from the proposed migration is at least 10% in the next time interval T^β .

$$T_{j,k}^{A \leftrightarrow B} > T^{Thresh} \quad (16)$$

Equation 16 forms the basis of the migration decisions. If we have more than two sub-clusters in the compute farm, then the sub-clusters which are expected to post highest savings are the potential candidates.

5.7 Migration Decisions

At any given time, a compute farm may have a number of sub-clusters with a number of active jobs. In order to make migration decisions that improve the throughput of the compute farm, we equate all the active jobs against each other; except for the ones which are on the same sub-cluster. The left hand side of Equation 16 is used to develop a comprehensive list. The list is sorted in descending order, which bring the jobs with highest savings on the top. We then traverse through the list in order and migrate the jobs if the qualify for final migration check i.e. enough resources are available on both the sub-clusters to entertain the swapped jobs.

We have identified five possible migration scenarios as shown in Figure 2 to Figure 5. In all the figures, each box represents a VM (compute node or a process belonging to a particular parallel application). Jobs j, k, L are shown as blue, red and green boxes respectively. The white boxes represent free nodes, i.e. nodes that are currently not processing any job. The grey boxes show a job that is not part of the migration sequence.

Each scenario is discussed as follows:

1. Migration of equal nodes: This is a straight forward migration scenario where $\eta(j) = \eta(k)$. The VMs are simply swapped across the sub-clusters. An example scenario is shown in Figure 2.
2. Migration to free nodes: This is another straight forward migration scenario which job ' j ' is migrated to a cluster (e.g. sub-cluster 'B') which has enough free nodes to entertain the migration as shown in Figure 3. The free nodes are migrated to sub-cluster 'A' to maintain the total number of nodes available for computation.

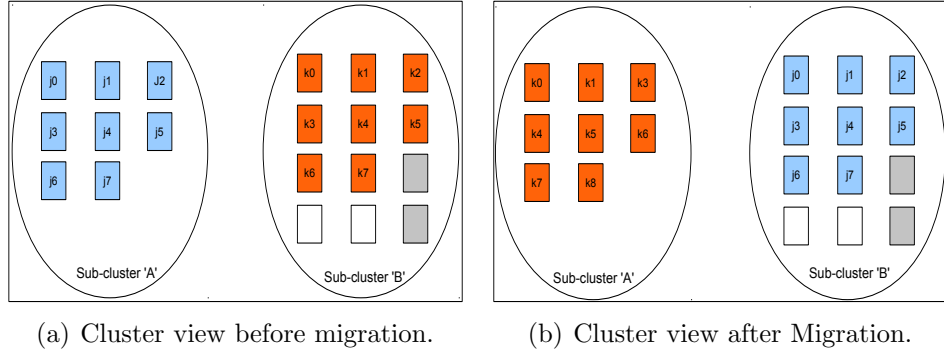


Figure 2: Migration decision scenario 1 - Migration of equal nodes.

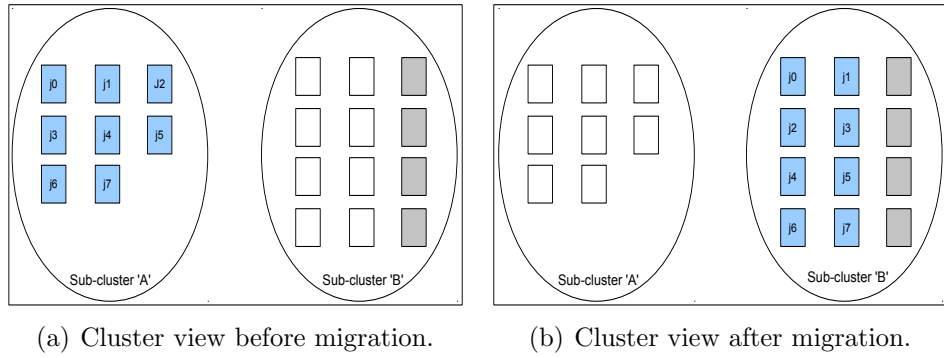


Figure 3: Migration decision scenario 2 - Migration to free nodes.

3. Migration with free nodes: This is a slightly complex migration case where $\eta(j) > \eta(k)$. The jobs (j and k) are migrated only if there are enough free nodes on the sub-cluster entertaining the job ' k '. i.e. $\eta(j) \leq \eta(k) + \zeta(k)$, where $\zeta(k)$ gives the total number of free nodes on sub-cluster entertaining job k . An example scenario is presented in Figure 4. The free nodes from the cluster 'B' are also migrated to the cluster 'A'.
4. Migration with multiple jobs and free nodes: This is a complex scenario, where $\eta(j) > \eta(k) + \zeta(k)$. In this case we traverse the sorted list and look for jobs in the sub-cluster 'B' such that $\eta(j) \leq \eta(k) + \zeta(k) + \sum_{\gamma} \eta(\gamma)$, where γ represents the jobs on sub-cluster entertaining job k . The migration decision is made only if the threshold condition is met. An example is shown in Figure 5. In this scenario we assume the total time penalty to migrate jobs k and L to cluster 'A' is less than the total time gained if job j is migrated to cluster 'B'. To maintain the symmetry, the free nodes are also migrated from cluster 'B' to cluster 'A'.
5. Migration due to insufficient memory: As discussed in Section 5.4, the swap partition thrashing is the most costly operation. We do not predict the performance for this case. In the case an application is thrashing, we simply migrate it to the first available sub-cluster which has sufficient

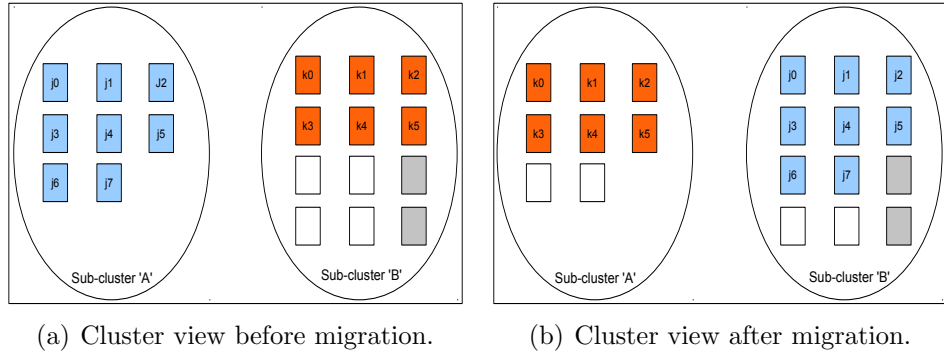


Figure 4: Migration decision scenario 3 - Migration with the free nodes.

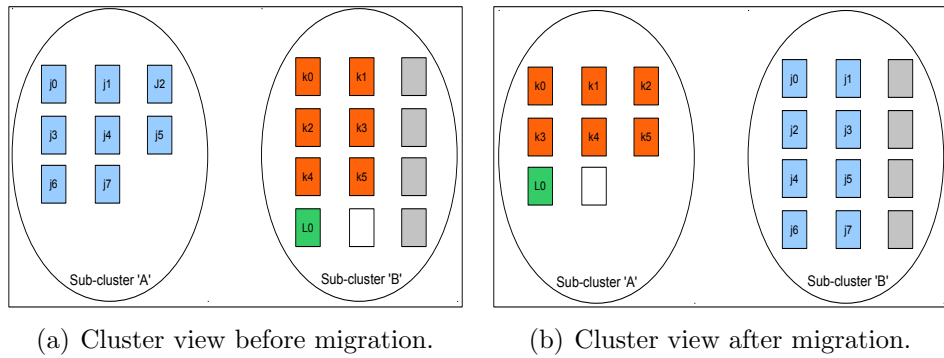


Figure 5: Migration decision scenario 4 - Migration with multiple jobs.

resources to entertain the job.

To ensure framework benefits from the migration decisions, a migrated job has to wait for $\beta\tau$ seconds to be considered for another migration.

6 Framework Implementation

The framework requires a runtime engine to generate the computation and communication profiles of the active jobs. The high level diagram of the ARRIVE-F is presented in Figure 6

The user submits the job through the ‘*job submit*’ routine. This routine simply inserts the job in the database along with its path and environment information.

The job queue is continuously analyzed by the *job scheduler* and the jobs are dispatched to the respective clusters based on the easy backfill algorithm. Details of scheduling are provided in Section 6.1.

Each compute node has a ‘*local resource manager*’ daemon (LRM) responsible for updating the node statistics like memory utilization and the communication network utilization. The memory utilization is done through the `/proc` file system, whereas, the communication network utilization is done through the PMPI

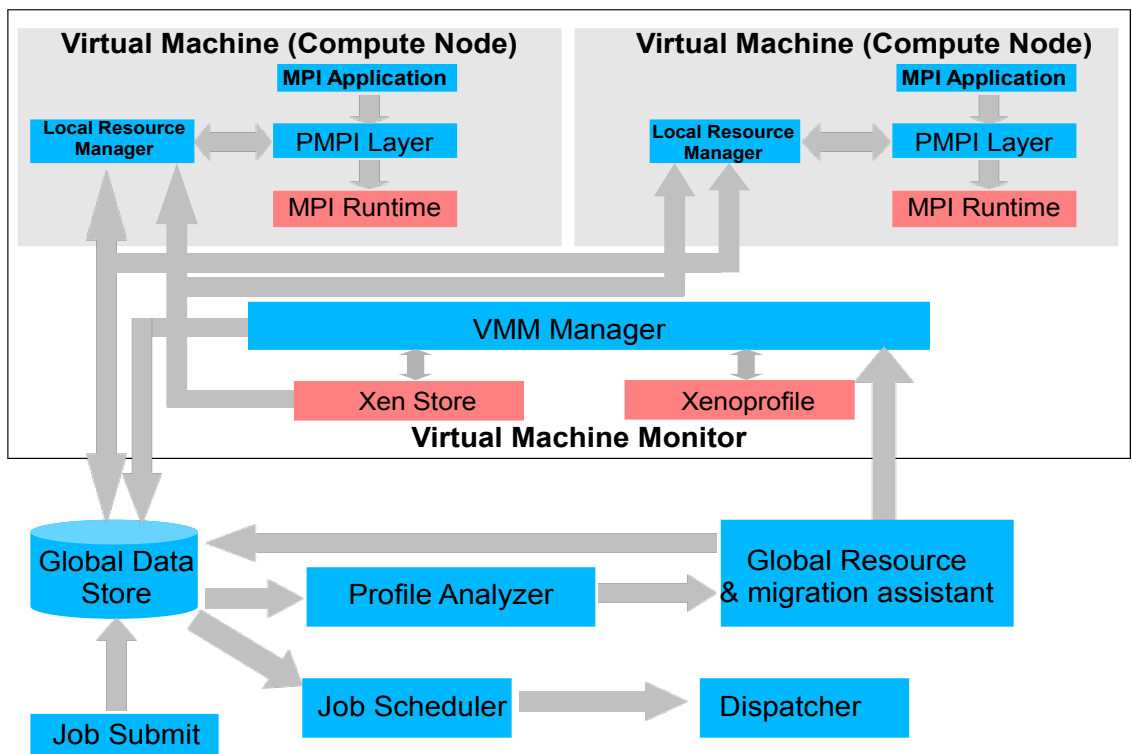


Figure 6: ARRIVE-F block view.

layer. In order to achieve lightweight MPI message profile, the LRM shares memory with the PMPI layer. The PMPI layer logs every distinct message (in terms of message size, communication destination, and message type). The distinct messages are held in an array for fast processing. Currently our implementation only entertains a subset of MPI primitives, namely MPI_Send, MPI_Recv, MPI_Alltoall, MPI_Isend, MPI_Irecv, MPI_Wait, MPI_Wait_all. The LRM periodically updates the summary of these statistics to the global data store which is a MySQL database. The LRM has an interface to the Xenstore database to retrieve the current VMM host of the compute node and update the database.

The memory profile of each process is obtained through the `/proc` file system. The local resource manager, periodically reads the `/proc` file system and determines the total RAM and swap space utilization. If the compute node is utilizing the swap space, then the amount of swap space utilized by the compute node plus the total physical memory (RAM) of the node gives the required RAM by the node. The framework tries to give the VM more memory by adjusting memory within the VMM. If the framework is unable to free the required amount of RAM, it looks for a sub-cluster which has available RAM equal to SWAP partition utilization + physical RAM + 10% tolerance for the overhead and the future needs of the process. The VM (or the parallel job) is immediately migrated to the first VMM that can entertain the requirements.

The CPU profiles are generated through the passive domain profiling provided by the Xenoprofile [18], which is the only option available to read the hardware performance counter data under Xen. OProfile is a system-wide profiler for Linux systems that leverages the hardware performance counters of the CPU to enable profiling of a wide variety of application statistics like floating point operations, L1/L2 cache misses, L1/L2 TLB misses etc. OProfile is a statistical continuous profiler where, profiles are generated by regularly sampling the performance counter registers on each CPU through an interrupt handler. Once the performance counter register reaches a certain threshold value (called event count), an interrupt is generated which is handled by oprofile kernel module. The module then saves the program counter (PC) value and maps the PC value to an application/library. The CPU profiles are read by the ‘*VMM Manager*’ daemon (VMMD) which resides in the domain-0. Like the LRM, the VMMD periodically sends data to the global data store. To minimize the profile overhead, the event counter threshold is kept at 500K for all the events and the sample directory is kept in the RAMFS. It may be noted that decreasing the event counter threshold results in improved prediction accuracy at the cost of increased profile overhead time. Xenoprofile does not allow multiplexing of hardware events; therefore the VMMD is capable of manually switching between the events. The VMMD also listens for the messages by the migration assistant to ensure proper migration of the compute nodes. Both daemons (LRM and VMMD) also provide a command line interface for the management purposes.

The ‘*profile analyzer*’ is responsible for predicting the job completion times and is based on the CPU and communication models explained in Sections 5.2

and 5.3. The Profile analyzer pulls the information from the MYSQL database after a given time interval. It then analyzes the jobs which have sufficient runtime left as defined in the previous sections. Once it establishes that the migration will result in an improved turnaround time of the involved job (or jobs), it then signals the *migration assistant* to take necessary action.

Migration assistant performs the migrations sequentially, e.g. First, a VM is migrated from sub-cluster ‘A’ to sub-cluster ‘B’. In the next sequence a VM from sub-cluster ‘B’ is migrated to sub-cluster ‘A’, as shown in Figure 7. This is due to the limited memory on the VMMs. Migrating in sequence ensures the no VMM runs out of memory.

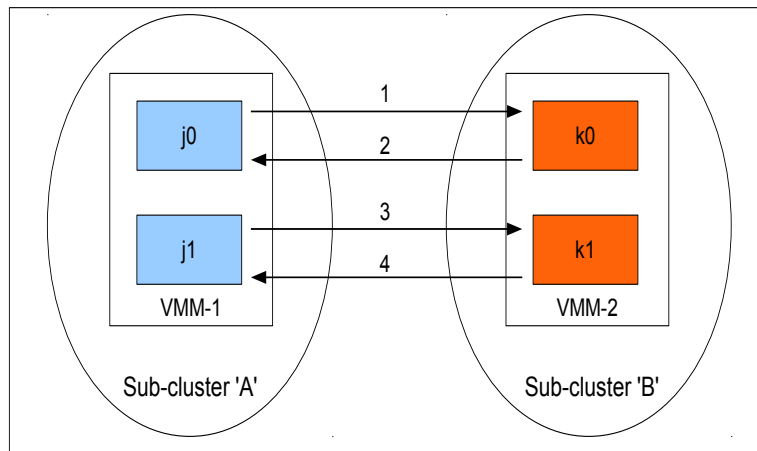


Figure 7: Migration Sequence of a cluster exchange between two parallel applications.

It may be noted that during the migration, xenoprofile daemon is de-initialized and no profile is maintained for the VMM at that time. This is due to limitation of xenoprofile which tends to crash the whole VMM if a VM being profiled is migrated. This limitation does not affect the system as the profiled data during migration in any case is not valid due to the noise introduced by domain-0.

We also provide a dashboard called the ‘*control center*’. The control center gives a live cluster view the HC providing useful information such as the available nodes, the jobs in the scheduler queue and the jobs that have been dispatched along with their estimated and elapsed times.

The framework also requires static cluster information, which is also stored in the database. The static cluster information is discussed in Section 7.3.1

ARRIVE-F is licensed under GPL version 3, and its source code can be downloaded from <http://cs.anu.edu.au/Muhammad.Atif/opensource/arrivef>.

6.1 Virtualization-aware Scheduler

For this research, we want a scheduler that is virtualization-aware and can dispatch jobs to a heterogeneous compute farm that is made up of smaller homoge-

neous sub-clusters. At the time of research there was no open source cluster scheduler that met our requirements; therefore we developed our own virtualization-aware scheduler from ground up.

Our scheduler is aware of the fact that the compute nodes are VMs, which belong to various distinct sub-clusters at any given time and that the VMs can ‘migrate’ between multiple sub-clusters in the compute farm if required. This requirement is not essential in a homogeneous compute farm where VM migration is not desired. The scheduling algorithm is based on an easy backfill algorithm with three major changes discussed as under.

1. No job is dispatched among the sub-clusters: As discussed above, we deal with heterogeneity by dividing an HC into a number of homogeneous sub-clusters. The scheduler keeps track of the number of free nodes in each of the sub-cluster and a job is dispatched to a sub-cluster as a whole.
2. Backfill algorithm for the sub-clusters: Our scheduler is based on the EASY backfill algorithm, We modified the backfill algorithm to enable backfilling of jobs across multiple clusters. When a job at the head of the queue cannot be dispatched, the time reservation of the job is done and *out-of-order* jobs are dispatched to the sub-clusters. These out-of-order jobs must have the expected runtime within the bounds of the completion time of the first job that will release enough resources (the compute nodes) to enable the dispatcher to launch the *head job* on *any* sub-cluster.
3. Scheduler lock during migration: When the profile analyzer and migration assistant are performing migration of the compute nodes, the scheduler is prevented from dispatching jobs on the involved sub-clusters. If the dispatcher and the migration assistant are not synchronized, a process belonging to a parallel job might get dispatched to a ‘free’ compute node that is part of a migration sequence. This will result in a job running across multiple sub-clusters. Therefore, once the profile analyzer signals the migration assistant about a migration decision, the dispatcher is locked from dispatching jobs. The locking is carried out with the assistance of MySQL locking primitives.

It may be noted that the scheduling algorithm is not fundamental to this research. The framework is developed independent of the scheduling methodology and can work with any batch scheduling algorithm as long as it is able to provide VM host information and the necessary locks.

7 ARRIVE-F Accuracy and Overheads

7.1 Experimental Setup

For the experiments we used Xen 3.3.0 compiled from source using GCC 4.2.4. The hypervisor is patched with the live migration optimization [8]. The domain

0 kernel is a Xenon-linux 2.6.31.12 kernel. Each domain 0 has three network cards, one for management and two for the VMs. Each VM is provided with one CPU and two gigabit ethernet network interfaces. The VMs utilize the ‘multiple shared bridge’ configuration. As concluded in [7], this configuration gives the best results under OMPI. The VMs in sub-cluster ‘C’ utilize 100 Mbps network interfaces to provide heterogeneity in communication. All the VMs run Ubuntu Intrepid 8.10 and are mounted through the network file system (NFS) to enable migration. The experimental cluster consists of a number of sub-clusters as given in Table 1.

Table 1: Heterogeneous compute farm

Cluster	CPU Type	Memory	Total Machines
A	4 × Opteron 2.2 Ghz	4 GB	2
B	4 × Phenom II 3.0 Ghz	4 GB	2
C	4 × Phenom II 3.0 Ghz	4 GB	2
D	2 × Athlon 2.0 Ghz	1.2 GB	4

7.2 Applications

We utilize the NAS Parallel Benchmarks [3] and the High Performance Linpack (HPL) [15] to validate the framework.

The NAS Parallel Benchmarks (NPB), developed and maintained by NASA Advanced Supercomputing (NAS) Division are a small set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications.

We have used only a subset of these benchmarks namely MG, GC, FT, IS, LU and EP.

The reason for not including the I/O intensive benchmarks like BTIO is; the migration of VM in Xen requires the VM file-system to be mounted on a shared file system visible to both (source and destination) the virtual machine monitors. Our experimental platform has a slower NFS server. As the I/O intensive applications write to the file system, this will result in poor performance for the I/O intensive jobs. This will result in the I/O intensive job always getting migrated to the slowest cluster.

All the benchmarks are compiled with GCC 4.3.2 with `amdfam10` optimization. This optimization enables SSE3 and is backward compatible with AMD K8 architecture.

7.3 Comparison of Predicted and Actual Execution Times

In this section, we present the comparison of the actual execution times with the predicted times as generated by the framework. For the ease of comparison, we present two tables: one for the computational model accuracy and the other for the communication model accuracy. We have used NPB class ‘B’ for these experiments as their wall clock time is higher than class ‘A’.

7.3.1 Static Cluster Information

The framework requires static cluster information in order to predict the estimated time of an application.

As evident from Equation 2, the framework needs the total number of CPU cycles required to service individual micro-architectural events like floating point operations, L1 and L2 cache misses. The total number of cycles required to service cache misses are determined by the BIOS and kernel developer guide’s provided at the manufacturers site [2].

The Flop penalty is calculated by the “flops” benchmark by Al Aburto [5]. The benchmark performs eight distinct module runs using different combinations of floating point operations (FADD,FSUB,FMUL,FDIV). The number of benchmark runs are then combined to present mega-flops (MFLOPS). We have taken MFLOPS(3) value to compute the average number of cycles for each machine which considers 3.4% FDIV. The MFLOPS(1) and MFLOPS(2) take relatively high number of FDIV (9.6% and 9.2%), an assumption that has been criticized in the original source code. Whereas MFLOPS(4) does not consider the division operation at all, which again is not a real world scenario. The C program was compiled with `-march=amdfam10 -O3`. It may be noted that theoretical floating point operations per cycle of machines are different and not considered.

The bandwidth and latency information is captured through the OSU benchmark suite [4]. OSU benchmark suite contains a number of micro-benchmarks to test an MPI-2 compliant cluster for latency and bandwidth. We have used Multi-pair latency and bandwidth benchmarks to obtain the individual cluster latency (e.g. $l_A(s), l_B(s)$) data. The cluster-wise latency and bandwidth results are stored in the MYSQL database, which can be retrieved by ARRIVE-F while developing communication profiles. Unlike LogP and its variants, we do not use the overhead parameter. The OSU benchmarks essentially capture the overheads while testing the system for latency and bandwidth.

The OSU benchmarks capture latency and bandwidth information at a given number of message sizes (2^i , where $i = \{1,2,3 \dots 32\}$). To obtain latency and bandwidth information for a message size not available in the database, we map the results to the nearest message size. In case the message size is greater than 2^{32} (4 MB), we use a linear extrapolation.

7.3.2 Accuracy of the CPU Model

For the CPU model, we utilize sub-clusters ‘A’ and ‘B’, which have similar network but different CPU architecture. We execute the benchmark on cluster ‘A’ and the framework predicts the wall clock time for the cluster ‘B’.

The columns T_A^{act} and T_B^{act} give the time taken by the benchmark on the respective sub-clusters. The column $T_{A \rightarrow B}^{\text{act}}$ gives the wallclock time predicted by the framework for the cluster ‘B’ based on the data collected from the cluster ‘A’ and is calculated from Equation 10. The use of T_A^{act} essentially means that the user time estimate is exactly equal to the runtime of the application. The column ‘%

Table 2: Computational model accuracy (predicted vs actual)

Benchmark	T_A^{act}	T_B^{act}	$T_{A \rightarrow B}^{act}$	% Acc CPU	% Acc Prof
CG.B.8	104.5	57.9	71.2	75.5	81.3
FT.B.8	98.2	87.6	79.3	88.6	90.5
LU.B.8	199.7	81.3	107.1	46.0	76.0
HPL.N15K	150.7	62.2	68.5	56.2	90.8

Acc CPU’ shows the percentage accuracy of the execution time projection based on linear CPU frequency which forms the basis of previous research works. The CPU frequency speedup ratio is calculated by f_B/f_A . The column ‘% Acc Prof’ gives the execution time percentage accuracy of our framework. It is clear that our framework consistently outperforms the CPU frequency method.

Example Calculations

In this section, we show the sample calculations that lead to the performance estimation for the FT.B.8 and LU.B.8 benchmarks.

Table 3 gives the performance penalty of sub-clusters in terms of CPU cycles.

Table 3: Penalty in CPU cycles

Cluster	Freq	Penalty in CPU Cycles		
		Flop	L1 Cache Miss	L2 Cache Miss
A	2.2 GHz	2.5	15	108
B	3.0 GHz	1.25	15	108

for the said benchmarks. The column *Freq* gives the CPU frequency of the compute machines in the relevant sub-clusters. The columns *Flop*, *L1 Cache Miss* and *L2 Cache Miss* give the number of the CPU cycles required to service each operation.

Table 4: Calculation of time estimate (seconds) based on computation model $\tau = 50$ seconds

Benchmark	Description	FPU	L1	L2
FT.B.8	Pctr Events	20251	1118	180
	Base sub-cluster ‘A’ time	11.5	3.8	4.4
	Predicted sub-cluster ‘B’ time	4.2	2.8	3.2
LU.B.8	Pctr Events	56771	1038	285
	Base sub-cluster ‘A’ time	32.2	3.5	7.0
	Predicted sub-cluster ‘B’ time	11.8	2.6	5.1

The performance counter data and the associated cost (actual and predicted) is shown in Table 4

Pctr Events represent the performance counter events. The event count for each counter is set to 500K. *Base sub-cluster ‘A’ time* gives the time penalties for

the given performance counter on sub-cluster ‘A’. *Predicted sub-cluster ‘B’ time* gives the predicted time penalties for the given *Pctr Events*. From the equations 3 and 9, we can compute the potential time savings for $\tau = 50$ as follows:

$$t_{A \rightarrow B, FT.B.8} = (11.5 - 4.2) + (3.8 - 2.8) + (4.4 - 3.2) = 9.47 \quad (17)$$

The total potential time saved for the entire duration of the benchmark can therefore be calculated through Equation 10:

$$T_{A \rightarrow B, FT.B.8} = 9.47 \times \frac{98.2}{50} = 18.6 \quad (18)$$

This means that the FT.B.8 benchmark should execute in $T_A^{\text{act}} - 18.6 = 79.4$ seconds. The actual time of the benchmark on the sub-cluster ‘B’ (T_B^{act}) was 87.6 seconds. This gives 90.5% accuracy of our prediction.

Similarly, the total time saved by LU benchmark if executed on sub-cluster ‘B’ is $T_{A \rightarrow B, LU.B.8} = 92.6$ seconds. The runtime of the benchmark is thus $T_A^{\text{act}} - 92.6 = 107.1$ seconds. The actual time taken by the LU.B.8 benchmark on cluster ‘B’ is 81.3 seconds. This gives a prediction accuracy of 76%.

7.3.3 Comparison of Communication Model

For the communication profile, we utilized the sub-clusters ‘B’ and ‘C’. The only difference between these two clusters is the communication link. We are only displaying the results from sub-cluster ‘C’ to ‘B’. The results from the sub-clusters ‘B’ to ‘C’ are similar. Each result has a standard deviation of $\pm 4\%$.

Table 5: Communication model accuracy (predicted vs actual)

Benchmark	T_C^{act}	T_B^{act}	$T_{C \rightarrow B}^{\text{act}}$	% Acc Prof
CG.B.8	141.0	57.9	66.2	88.8
FT.B.8	375.1	87.6	90.15	97.2
LU.B.8	106.8	81.3	74.38	91.5
HPL.N15K	150.7	62.2	80.2	71.1

It can be seen from Table 5 that the accuracy of the communication model decreases for the application which sends or receives a high frequency of messages. This is because Xen uses a split driver interface which is highly CPU intensive [7]. For the applications like LU, which send high numbers of messages of small size, the CPU load due to processing on domain-0 is high, and this skews the projections. One can introduce domain-0 profiles in the framework to overcome this issue.

Example Calculations

In this section, we show the sample calculations that lead to the performance estimation for the FT.B.8 and LU.B.8 benchmarks from the sub-cluster ‘C’ to

sub-cluster ‘B’. It may be noted that both the benchmarks use the blocking communication. LU.B.8 utilizes high number of messages of relatively small size. This makes the benchmark latency driven. FT.B.8 utilizes MPI_Alltoall collective and the message size is in the order of MB.

Table 6: Communication calculations for $\tau = 50$ seconds

Benchmark	Msg Size (s)	Freq (n^B)	Latency at Msg Size ($l(s)$)		Time for messages	
			cluster ‘C’ ($l_C(s)$)	cluster ‘B’ ($l_B(s)$)	t_C^B	t_B^B
FT.B.8	8.39 MB	57.83	0.73	0.08	42.21	4.70
LU.B.8	1.79 KB	46288	0.00019	0.00013	15.03	10.28
	0.25 MB	471	0.012	0.00136	11.69	1.26

Table 6 gives the communication summary for the profile time τ . The column *Msg Size* gives the size of distinct messages sent/received by the benchmark during the profile period (τ), i.e. parameter ‘ s ’ in Equation 5. The column *Freq* represents the number of the distinct messages (represented by *Msg Size*) that were transmitted or received during the profile period, i.e. ‘ n_j^B ’, where j is either FT.B.8 or LU.B.8. The column *latency (s) at msg size* gives the latency of the two networks at the given message size, i.e. variable ‘ $l(s)$ ’ in Equation 5. The column *Time for messages* gives the communication times for the messages for each of the cluster for the given message size. For cluster ‘C’, this time is computed by Equation 5, whereas for cluster ‘B’, we utilize Equation 6.

The potential time saved if FT.B.8 benchmark is moved from the sub-cluster ‘C’ to sub-cluster ‘B’ is calculated from the equations 3 and 9 as follows :

$$t_{A \rightarrow B, FT.B.8} = 42.21 - 4.70 = 37.51 \quad (19)$$

Here, the CPU component is 0 as both the sub-clusters have identical hardware specifications. The total potential time saved for the entire duration of the benchmark can therefore be calculated through Equation 10:

$$T_{A \rightarrow B, FT.B.8} = 37.51 \times \frac{375}{50} = 281.3 \quad (20)$$

Equation 20 suggests that the FT.B.8 benchmark should execute in 375 - 281.3 = 93.7 seconds, compared to actual time of 87.6 seconds. The prediction accuracy in this case is 93.5%.

Similarly, the predicted time for the LU.B.8 benchmark ($T_{C \rightarrow B}$) is 74.38 seconds. This gives us the prediction accuracy of 91.5%.

7.4 Framework Overheads

Figure 8 gives the comparison of the relative wall clock times for a set of NPB class ‘B’ and HPL benchmarks with and without the prediction framework. Approximately 1.2% of the profile overhead is due to Xenoprofile. The communication

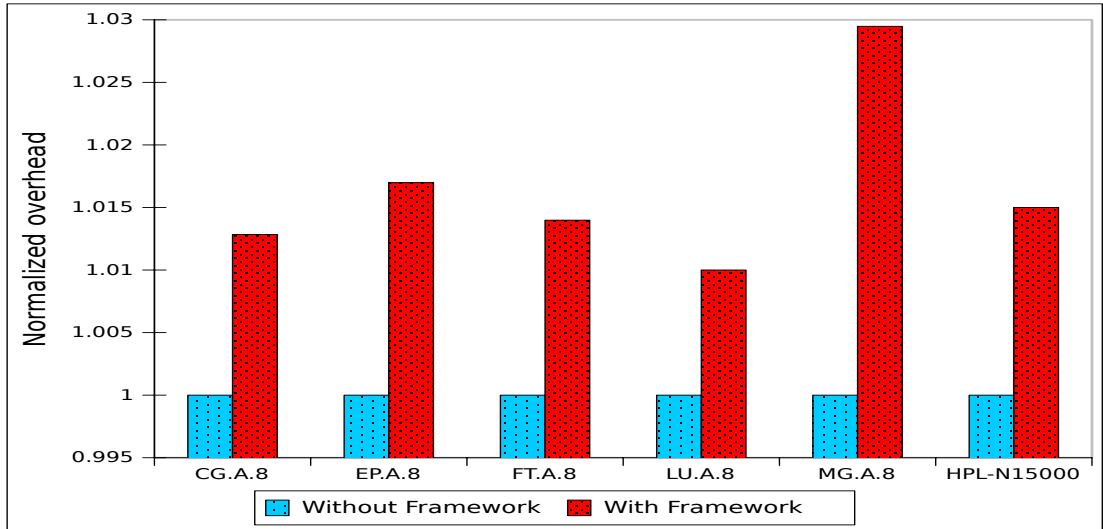


Figure 8: Overheads of the profiling framework and daemons.

profiles have an overhead of 0.3% to 0.7% depending upon the communication rate of the benchmark, except for the MG benchmark. The MG benchmark uses non-blocking messages and the overhead is due to the retrieval of the corresponding non-blocking requests, as discussed in Section 5.3. These overheads include the cost of updating the database after every τ seconds.

7.5 Live Migration Experiments

In this section, we present simple test cases in a controlled environment where our framework is able to detect the best suited hardware and migrate the jobs to save time. 8, we will show the throughput improvement experiments. The results of the workload data representative of the real world are presented in Section 8.

7.5.1 Computational Requirement Migration

In this scenario, we present the results of the case where jobs are swapped between the two sub-clusters based on the CPU requirements. We use FT.B.8 and LU.B.8 benchmarks from the NPB suite. As each VM is sharing the same GigE interface of domain-0, the bandwidth becomes the limiting factor for FT.B.8. Even if the benchmark is moved to a CPU with higher clock rate and better floating point operations per cycle, it does not benefit. LU.B.8 is a floating point intensive benchmark and bandwidth is not the limiting factor therefore; it benefits from the hardware with a higher floating point rate.

For this experiment, LU.B.8.1500 was dispatched to cluster ‘A’ whereas, FT.B.8.300 was scheduled on cluster ‘B’. Both the sub-clusters have similar network but different CPU architectures as detailed in Table 1.

The framework was able to determine the potential time saved if the jobs

were swapped among the clusters. Table 7 shows the results of the experiment in summarized form.

Table 7: Computational Requirement Migration

Benchmark	Sub-cluster	T^{act}	$T_{A \leftrightarrow B}^{\text{act}}$	Time saved
LU.B.8.1500	A	1299	659	601
FT.B.8.300	B	1263	1302	

The column ‘Sub-cluster’ gives the initial sub-cluster allocated to jobs. T^{act} gives the wall clock time of the jobs on the sub-clusters in the base run where no migration took place. $T_{A \leftrightarrow B}^{\text{act}}$ gives the wall clock time of the jobs in the migration run. The framework is able to save approximately 601 seconds per process in this case.

7.5.2 Communication Requirement Migration

In this scenario, we test the ability of the framework to detect and make the migration decision based upon network utilization of the MPI jobs. We use LU.A.8 and FT.A.8 benchmarks from the NPB suite. LU benchmark is latency-driven, where each process sends or receives a large number of messages of relatively small size (typically less than 0.28 MB). FT benchmark is bandwidth driven and uses an MPI_All2all communication. The message size for the FT class ‘A’ benchmark is approximately 2 MB.

We run the experiments on sub-clusters ‘B’ and ‘C’. Both the clusters have similar CPU, but different network interfaces. Here, FT.A.8 benchmark was dispatched to sub-cluster ‘B’ and LU.A.8 was allocated to sub-cluster ‘C’. The framework detects the communication heterogeneity between the two clusters, and migrates the jobs accordingly. Table 8 shows the results in the summarized form.

Table 8: Communication Requirement Migration

Benchmark	Sub-cluster	T^{act}	$T_{B \leftrightarrow C}^{\text{act}}$	Time saved
FT.A.8.925	B	3387	1104	1624
LU.A.8.13367	C	1170	1829	

7.5.3 Migration due to Memory - Swap Partition

In this case, we utilize sub-clusters ‘A’ and ‘D’. The sub-cluster ‘A’ is already executing FT.A.4 when HPL (N=15500) was dispatched to the sub-cluster ‘D’.

At $N=15500$, a 4 process HPL benchmark starts to thrash the swap partition on a 512 MB compute node. The framework is able to detect the thrashing and work out that ‘D’ cluster cannot make more RAM available to the job. Therefore, the job is swapped with the first available compute node that can not only support the required memory, but the job running on it has memory requirements that can be satisfied by cluster ‘D’.

A summary of the test results is provided in Table 9. In this particular case, framework was able to save 408 seconds. These times can be significantly high if the benchmark run was in the order of hours.

Table 9: Memory Requirement Migration

Benchmark	Sub-Cluster	T^{act}	$T_{A \leftrightarrow D}^{\text{act}}$	Time saved
HPL-N15.5K	D	1089	370	408
FT.A.4.280	A	416	727	

8 Compute Farm Throughput

To determine the throughput of the compute farm with the resource relocation framework, the scheduler is supplied with the workload which is representative of real world data. We then compare it against the base run using the same scheduler but without the profiling framework active. In both the experimental runs the scheduler is supplied with the same stream of rigid jobs with the approximate expected runtime given for each job. Rigid jobs are the parallel applications that specify the required number of processors at the time of submission [17]. A large body of models exist for the generation of workload; among them Downey’97 [13], Lublin-Feitelson [17] and Tsafrir-Feitelson [21] are worth mentioning.

We used the Lublin-Feitelson model [17] to generate the job queue. The Lublin-Feitelson is a very detailed model for rigid jobs that includes an arrival pattern with a daily cycle and runtimes that are correlated with the number of nodes. The model was modified to remove the occurrence of the interactive jobs. The job size was limited to a maximum of 8 processes due to hardware limitations. The Lublin-Feitelson model produces a single stream of jobs for an entire compute cluster. As we have four distinct sub-clusters, each of which can entertain maximum of eight process parallel job, four job streams were created and merged to form a unified stream of jobs. The number of jobs was arbitrarily chosen to keep the experiment runtime to approx 3 hours. The list of jobs generated by the model was randomly allocated to one of the NPB kernels namely; the CG, EP, FT, IS, LU, MG benchmarks. The number of iterations of these benchmarks were changed to match the approximate expected runtime provided by the Lublin-Feitelson model. This was done by calculating an iteration size of

each benchmark on cluster ‘A’. Therefore the user time estimates are based on sub-cluster ‘A’.

It may be noted that improvement in the compute farm throughput is highly circumstantial in nature and depends on the characteristics of the jobs in the queue. We have conducted several experiments with different streams, only three experiments are shown in the thesis. The results of the two experiments not presented in the thesis were similar. We produced a separate job stream for each experiment. The job streams are presented in Appendix 10

For these experiments, the value of $\tau = 50$ seconds and $\beta = 20$ time blocks. This means that we only migrated applications that have an estimated runtime of over 1000 seconds. The profile analyzer was activated after every 4τ i.e. every 200 seconds. The Threshold value (T^{Thresh}) is $0.1\beta\tau\eta$, where η is the total number of processes of the jobs involved in the migration decision. We use the throughput, the average waiting time and the average turnaround time to show the percentage improvement of the migration run over the base run.

The ‘throughput’ is defined as the number of processes that complete their execution per time unit. We calculate the relative throughput by determining the total number of jobs completed by the base-run in the time that took migration run to complete the execution of all the submitted jobs. The ‘wait time’ is defined as the time a job has to wait in the queue before getting served (or dispatched). The wait time is computed by subtracting the submit time of each job from its dispatch time. We then take the average of the wait times of all the jobs in the queue. The ‘turnaround time’ is defined as the interval between the submission of a job and its completion. The turnaround time is obtained by subtracting the submit time from the job finish time. Similar to the wait time, we take the average of the turnaround times of all the jobs in the queue. This time has an implicit parameter of average execution time.

8.1 Experiment Number 1

In this experiment a stream of 330 jobs was given to the scheduler. The number of jobs was arbitrarily chosen to keep the experiment runtime to approx 3 hours. The job stream is provided in Appendix 10.1. The inter-arrival-time of the jobs was selected to represent a normal rush hour.

The Lublin-Feitelson model also generated jobs that required 3/5/6 and 7 processes. Compiling the NPB for these number of nodes is not possible except for the EP benchmark. Therefore, the number of processors for these jobs was scaled up in case their runtime was not approximately equal to the EP kernel runtime on cluster 1.

The scenario was tested three times and results of each run are presented in Table 10. The columns ‘Base-run’ and ‘Migration Run’ give the statistics of the respective runs. The columns ‘W.Time’ and ‘TA.Time’ represent the average wait time and the average turnaround time of jobs respectively. The column ‘Total time’ gives the total time, taken by the respective base-run or the migration run,

to complete the execution of all the 330 jobs supplied to the scheduler. The column ‘jobs @ mig.run’ gives the total number of jobs completed by the base-run in the time that took migration run to complete the execution of all the 330 jobs.

The average throughput improvement achieved for this particular case is 27%. The average time saved by the framework is 3104 seconds, which reflects an impressive time saving of 32%. Compared to base-run, the migration run reduced the average wait time for jobs by 55% and average turnaround time of the jobs was improved by 54%.

As a single second difference in the execution of any job can result in different sub-cluster allocations to the subsequent jobs, the total times are different for each experimental run.

Table 10: Base-run vs migration run

Sr. No	Base-run				Migration Run		
	W.Time	TA.Time	Total Time	Jobs@mig.run	W.Time	TA.Time	Total Time
1	4493	4654	12434	255	2747	2913	9740
2	4469	4684	12782	267	2961	3129	9380
3	4531	4749	12837	258	2982	3119	9619
<i>Avg.</i>	<i>4498</i>	<i>4696</i>	<i>12684</i>	<i>260</i>	<i>2896</i>	<i>3053</i>	<i>9580</i>

Two jobs that mainly contributed to slowing down the throughput of the compute farm in the base-run are given in Table 11. These are not the only jobs which contributed to the low throughput of the compute farm, as shown in the migration decisions made by ARRIVE-F below.

Table 11: Job allocation in base run

Job Name	Sub-cluster	T ^{est}	T ^{act} Base	Reason
FT.B.4.20	D	92	1148	Thrashing
IS.A.8.10655	C	2717	10728	Comm. Requirement

The column ‘Sub-cluster’ gives the initial sub-cluster allocation to the job. ‘T^{est}’ and ‘T^{act} Base’ give the estimated and actual times of the jobs.

For each experimental run of the migration-run, three distinct migration decisions were made by ARRIVE-F:

1. Migration 1: Like the base-run, in the migration run, FT.B.4.20 was allocated to Cluster ‘D’ which has 1.2 GB of physical memory per physical machine. FT.B.4 benchmark requires a minimum of 750 MB of physical memory to avoid thrashing and takes approximately 90 seconds to complete. As each VM on cluster ‘D’ can have a maximum of 512 MB of memory, the job used the swap partition. This resulted in the job taking

1148 seconds to complete. The migration framework was able to detect the case and the job was swapped with the job (FT.A.4.156) running on cluster ‘C’. The migration results are detailed in Table 12.

Table 12: Experiment 1: Job migration 1

Job Name	Sub-cluster	T^{est}	T^{act} Base	$T^{\text{act}}_{C \leftrightarrow D}$ Mig.	Time Saved
FT.B.4.20	D	92	1148 (D)	415	720
FT.A.4.156	C	230	95 (C)	108	

The column ‘ T^{est} ’ shows the estimated time generated by the Lublin-Feitelson model. ‘ T^{act} Base’ gives the actual time taken by the job to complete in the base run. The brackets contain the sub-cluster on which the job executed in the base-run. The column ‘ $T^{\text{act}}_{C \leftrightarrow D}$ Mig.’ gives the time taken by the jobs in the migration run.

- Migration 2: The second migration performed by the framework was the sub-cluster swap between MG.B.8.5132 and FT.B.8.506. Initially MG.B.8.5132 was allocated to sub-cluster ‘A’ and FT.B.8.506 was allocated to sub-cluster ‘B’. The FT.B.8 is a bandwidth bound benchmark and it does not benefit from the higher CPU clock rate due to limited communication capabilities of the two GigE interfaces (multiple shared bridge configuration). The MG.B.8 benefits from the faster FPU offered by the sub-cluster ‘B’. The framework was able to determine that the compute farm will benefit from the migration swap of these two jobs and proceeded accordingly. The impact of the migration is shown in Table 13. The total time saved by the compute farm with this migration is computed by adding the times of Base run and subtracting them from the times of Migration run i.e. $(2332 + 2005) - (1769 - 2258)$. The results are average of 3 experimental runs each. It may be noted that MG uses non-blocking sends and its overheads are approximately 3% when run under ARRIVE-F.

Table 13: Experiment 1: Job migration 2

Job Name	Sub-cluster	T^{est}	T^{act} Base	$T^{\text{act}}_{A \leftrightarrow B}$ Mig.	Time Saved
MG.B.8.5132	A	2697	2332 (A)	1769	310
FT.B.8.506	B	2174	2005 (B)	2258	

- Migration 3: The last migration was in fact a sequence of migrations and represents the complex migration scenario presented in Section 5.7. Here, the framework migrated LU.A.8.12334 from sub-cluster ‘C’ to sub-cluster ‘A’. As seen from the previous experiments, LU.A.8 has a lesser penalty on cluster ‘C’ compared to the other NPB kernels. However, the framework was able to co-locate all the processes of CG.B.4.2286, eliminating the inter-node communication through the slower network interface. This resulted in CG.B.4 benefiting from the faster CPUs. Similarly LU.A.1 and LU.B.1 benefited from the higher flop rate of sub-cluster ‘C’. LU.A.8.12334 lost

CPU time which was made up by the faster communication network of cluster ‘A’.

Table 14: Experiment 1: Job Migration 3

Job Name	Sub-cluster	T^{est}	T^{act} Base	$T_{A \leftrightarrow C}^{act}$ Mig.
CG.B.4.2286	A	3268	3408 (D)	2043
LU.A.1.7385	A	5869	5870 (A)	4161
LU.B.1.455	A	1500	1850 (A)	1447
LU.A.8.12334	C	1850	1058 (B)	1838

The migration decisions enabled the HC to save a total of 3,984 seconds (adding the times saved by the three migration routines). However, the total time saved by the HC was 3104 seconds. The difference is because of the time lost by the HC in the migration-run due to increased wall clock time of the jobs that executed on clusters that were less desirable than those used by the base-run. For example, FT.B.8.3 with an estimated time of 12 seconds got dispatched to sub-cluster ‘D’ in the base-run, but in the migration-run it was allocated to the sub-cluster ‘C’. The sub-cluster ‘C’ has the slowest network interconnects and due to a smaller time estimate, the job was not migrated to a sub-cluster with the faster interconnects. It took at an average 82 seconds to complete its execution on sub-cluster ‘C’ compared to 14 seconds (sub-cluster ‘D’) in the base-run.

The variation in total time taken by base-run and migration run is also due to same reasons. As mentioned, even a single second difference in the execution of an application can result in different cluster allocation to subsequent jobs.

To visually show how ARRIVE-F makes significant time gain over the base-run, a gantt chart view comparing the base and migration runs is presented in Figure 9. Due to the space limitations, the chart is not to exact scale and the parallel applications are shown with a minimum grain of 5 seconds. The x-axis of the chart gives the sub-clusters along with the nodes. The y-axis is time in the blocks of 5 seconds. The color coding for jobs is same (for both the graphs) for an easier comparison. The width of the job reflects the total number of processes and the length represents the time.

As seen from the graphs, the first two hundred seconds of both the experimental runs are exactly the same. Migration 1 started at $T=200$, when ARRIVE-F detected swap partition thrashing by FT.B.4.20. ARRIVE-F swapped FT.B.4.20 (shown in red color - initially on cluster ‘D’) with FT.A.4.156 (shown as salmon color - initially on cluster ‘C’). Because of the high network I/O due to thrashing, the migration time was high. During that time (approximately from $T=200$ to $T=405$) no job was dispatched to either of the cluster involved in the migration. Note that from this point onwards, the compute jobs were given different compute node assignments.

‘Migration 2’ took place at $T=615$, which resulted in speeding up the MG.B.8.5132 job. Here MG.B.8.5132 (shown in green Color) exchanged the sub-cluster with the FT.B.8.506 benchmark (shown in blue color). MG.B.8.5132 finished earlier

on sub-cluster ‘B’ making way for the smaller jobs that were waiting to be dispatched in the queue. As sub-cluster ‘B’ is the fastest cluster, these jobs finished quickly. This is reflected as a higher throughput of the compute farm. It may be noted that the profiler did not activate at ($T \approx 400$ seconds) as it was busy with Migration 1. This is due to the serial implementation of the profile analyzer and migration assistant. We have a further discussion on this in Section 9.

8.2 Experiment Number 2

We utilized the same methodology to generate the workload for this experiment as with the experiment number 1. However, during the random allocation FT.B.4.* benchmarks were removed from the random job allocation list to ensure that no benchmark thrashes the swap partition. A total of 212 jobs were generated and the same job list was submitted to the scheduler for each experimental run. The list of jobs with the arrival time is given in Appendix 10.2. The results of three distinct and comparable experimental runs are shown in Table 15

As opposed to the experiment in Section 8.1 where an IS job kept sub-cluster ‘C’ busy for a significant time in the base-run, the experiment presented here had LU.A.8.51600 ($T^{\text{est}} = 7740$ secs) allocated to sub-cluster ‘C’ for the most of the duration of the experiment. This resulted in factoring out the slow communication in sub-cluster ‘C’. In essence, this benchmark involves the computation aspects of ARRIVE-F.

Table 15: Experiment 2: Base-run vs migration run

Sr. No	Base-run				Migration Run		
	W.Time	TA.Time	Total Time	Jobs@mig.run	W.Time	TA.Time	Total Time
1	1971	2159	7730	201	1998	2163	7320
2	1872	2047	7451	207	1806	1970	6940
3	1814	2011	8195	201	1847	2021	7527
<i>Average</i>	<i>1886</i>	<i>2072</i>	<i>7792</i>	<i>203</i>	<i>1884</i>	<i>2051</i>	<i>7262</i>

This experimental run only had one migration, which was between LU.B.8.3088 and FT.B.8.1093. Interestingly most of the jobs got allocated to the preferred clusters in this experiment.

Table 16: Experiment 2: Job migration 1

Job Name	Sub-cluster	T^{est}	T^{act} Base	$T^{\text{act}}_{A \leftrightarrow B}$ Mig.	Time Saved
LU.B.8.3088	A	2177	2077(A)	1237	641
FT.B.8.1093	B	4697	4657(B)	4856	

The total time saved by migration of FT.B.8.1093 and LU.B.8.3088 benchmarks was 641 seconds. This is determined by subtracting $T^{\text{act}}_{A \leftrightarrow B}$ Mig. from T^{act} Base for each benchmark and adding the result. The total time saved by the compute cluster is 530 seconds.



(a) Experiment Number 1 - Base Run. (b) Experiment Number 1 - Migration Run.

Figure 9: Gantt chart of the first 2000 seconds of experiment number 1

The throughput improvement due to this benchmark is 4%, which is significant considering only one migration was performed in the whole run. This result suggests that ARRIVE-F under normal circumstances will not result in slowing down the compute farm as, even with one odd migration, it is able to counter its overheads. The time saved by the framework is 530 seconds, which reflects a time saving of 6.8%. Compared to the base-run, the migration run reduced the average wait time for jobs by 1% and average turn-around time of the jobs was improved by 7%.

8.3 Experiment Number 3

In this experiment we removed the sub-cluster ‘C’, which uses 100 mbps ethernet for the VM communication. This resulted in a compute farm with similar communication interfaces and without the predictable migration based on the slower network. The experimental data for this experiment was generated through Lublin-Feitelson model, as with the previous experiments, but this time we changed the arrival time between the jobs to reflect a lesser load. This was done to see the how ARRIVE-F behaves under such a load. The same list of 194 jobs was submitted to the scheduler for each experimental run. The job details are provided in Appendix 10.3.

This experimental run saw ARRIVE-F making three migration decisions, all based on computational requirements. The overall throughput was improved by 13% and total time saved was 3360 seconds, which reflects an improvement of 33%. The average waiting time was reduced by a very impressive 298% and the average turnaround time was improved by 230%. For this experiment, the average waiting time and the turnaround times are not in proportion with the throughput improvement as in the previous experiments. We have further discussion on this in Section 9. The summary of results is shown in Table 17.

Table 17: Experiment 3: Base-run vs migration run

Sr. No	Base-run				Migration Run		
	W.Time	TA.Time	Total Time	Jobs@mig.run	W.Time	TA.Time	Total Time
1	949	1190	13263	171	357	579	9743
2	1232	1500	13962	169	383	561	10406
3	1158	1412	13568	173	379	573	10563
<i>Average</i>	<i>1113</i>	<i>1367</i>	<i>13597</i>	<i>171</i>	<i>373</i>	<i>571</i>	<i>10237</i>

This experiment had three distinct migrations as discussed:

1. Migration 1: The first migration was between CG.B.2.1184 and LU.B.8.2279 benchmarks. At time T=1400; the sub-cluster ‘B’ was executing a two process job (CG.B.2.1184) with rest of the six compute nodes available. At the same time, an eight process job (LU.B.8.2279) was executing on sub-cluster

‘A’. It may be noted that both LU and CG are floating point intensive. However, the potential time gained by moving an eight process LU benchmark to a cluster with faster FPU is far more than the time lost by a two process CG.B job if the later is moved to a sub-cluster with a slower FPU. All ARRIVE-F makes the migration decision by weighting in the number of processes of each job, therefore it was able to compute the potential time saving if the sub-clusters of these two jobs were swapped. As shown in Table 18, this resulted in speeding up an eight process job almost by a factor of 2. The two process job in fact lost around 304 seconds compared to the base-run. The total time saved was 719 seconds.

Table 18 shows the results in summarized form.

Table 18: Experiment 3: Job migration 1

Job Name	Sub-cluster	T^{test}	T^{act} Base	$T^{\text{act}}_{\text{B} \leftrightarrow \text{D}}$ Mig.	Time Saved
CG.B.2.1184	B	1952	1272 (B)	1576	719
LU.B.8.2279	A	2005	1947 (A)	857	

- Migration 2: This migration is similar to the one discussed in Experiment number 1. The only difference is presence of a four process LU.B benchmark in place of an eight process LU benchmark. FT.B.8 is bandwidth limited and was migrated to a slower cluster to make way for the floating point intensive LU.B.4, despite the later having less number of processes. Note that FT.B.8.1093 executed on a sub-cluster ‘A’ in the base-run compared to sub-cluster ‘B’ in the migration run.

Table 19: Experiment 3: Job migration 2

Job Name	Sub-cluster	T^{test}	T^{act} Base	$T^{\text{act}}_{\text{B} \leftrightarrow \text{D}}$ Mig.	Time Saved
LU.B.4.1157	D	2174	1526(D)	948	744
FT.B.8.1093	B	4697	4986(A)	4820	

- Migration 3: This experiment saw an IS.B.8.2149 getting swapped with an LU.B.8.2255 benchmark. IS benchmark has almost zero floating point operations and it communication bound, whereas the LU benchmark is a floating point intensive application. Here IS.B.8.2149 was executing on sub-cluster ‘B’ which offers twice the rate of flops compared to the sub-cluster ‘D’. ARRIVE-F was able to migrate the jobs saving 235 seconds. The migration occurred late in the LU.B.8.2255 run, therefore the savings were not as significant compared to the other migration results.

Table 20: Experiment 3: Job migration 3

Job Name	Sub-cluster	T^{test}	T^{act} Base	$T^{\text{act}}_{\text{A} \leftrightarrow \text{B}}$ Mig.	Time Saved
LU.B.8.2255	D	1984	1696(D)	1354	235
IS.B.8.2149	B	1850	1760(A)	1867	

9 Discussion on ARRIVE-F Migration Decisions

Based on the experiments in the Section 8, we have made following observations.

The migration decisions in ARRIVE-F are ‘partially’ based on the user time estimates. The user time estimate is a requirement of backfill scheduling algorithms widely used in the production HPC clusters, therefore the requirement is not that strict. However, users are known to give wrong time estimates which may result in incorrect migration decisions, i.e. a migrated job ends too soon. In the worst case jobs being swapped between the sub-clusters may finish their execution earlier than $\beta\tau$ seconds, resulting in loss of time and hence the throughput. The experiments discussed in the previous section has a relatively accurate user predictions; therefore such an occurrence was not observed.

ARRIVE-F does not migrate jobs with a small estimated runtime $T^{\text{est}} \leq \beta\tau$. The jobs are migrated only if the framework is sure that migration will result in saving the time of the compute farm. If the application time estimate is less than $\beta\tau$, and it is running on a slower cluster, then it is not migrated. In the worst case, this results in a lost opportunity for ARRIVE-F to improve the throughput of the compute farm. An example is FT.A.8.X job, with a user runtime estimate of 800 seconds, dispatched to sub-cluster ‘C’. The job will take more than 3000 seconds to complete, but the framework is unable to re-locate the job due to its limitations. One can add heuristics and fine-tune the compute farm for such cases. However, we have kept the framework implementation conforming to the mathematical model presented above.

In a similar issue, jobs are only migrated if both of them have remaining time $> \beta\tau$. One can set migration decision to only consider the remaining time of one job and disregard the remaining time of the others. However, in such a case, jobs coming later in the queue might suffer. An example of such a case is LU.B.8 on sub-cluster ‘A’ with estimated time of 2000 seconds, swapped with an IS.B.8 job with estimated time of 200 seconds on sub-cluster ‘B’. The new mapping will result in speeding up the LU.B.8 but the jobs in the queue will get allocated to a slower sub-cluster ‘A’ as IS.B.8 job will finish first. For these reasons, ARRIVE-F is not aggressive in nature and only migrates jobs once it is absolutely sure of an improved throughput.

ARRIVE-F does not model the working set size of the parallel application based on L1 and L2 cache sizes. While making prediction, ARRIVE-F assumes that the application will have similar cache misses on the destination hardware. While this is true to most of the NPB kernels, the LU benchmark is an exception. We have empirically found (through performance counter data on different machines) that the LU application benchmark has a working set size of between 512 KB and 1024 KB. When it is migrated from a sub-cluster ‘D’ to cluster ‘A’, it posts a gain in performance which is not identified by ARRIVE-F.

ARRIVE-F dispatches the jobs to the sub-clusters according to the availability of the compute nodes. At the time of dispatch the jobs are tried to be co-located on a VMM if possible. If the parallel application spans multiple physical ma-

chines (VMMs) then, for a sub-cluster with the slower communication interfaces (e.g. sub-cluster ‘C’), this results in slowing down a bandwidth intensive parallel application as the messages are routed through the physical interface. However, in case the application is co-located, the processes utilize the Xen’s bridge infrastructure. One way to avoid a case where a bandwidth intensive application is dispatched across two distinct physical machines belonging to sub-cluster with slower network interconnect is to fine tune ARRIVE-F according to sub-clusters in a compute farm e.g. the scheduler can be tuned to dispatch jobs with maximum of 4 co-located processes on sub-cluster ‘C’ as each physical machine on sub-cluster ‘C’ has 4 CPUs. However, as shown in experiments, if the cluster is running a mix of latency and bandwidth driven applications, treating all the clusters on the basis of available compute nodes gives reasonable improvements.

The migration decisions in ARRIVE-F can further be optimized. One example is the case shown below from Experiment 3. Here, CG.B.2.1647 is running on the fastest cluster (sub-cluster ‘B’), as shown in Figure 10. The cluster ‘A’ has six nodes available for computation (i.e. nodes are idle) and the job on the head of the queue is an eight process job (LU.B.8.5) as shown in Table 21. If the scheduler migrates CG.B.2.1647 to sub-cluster ‘A’, the job at the head of queue can be dispatched. ARRIVE-F did not migrate the job because moving CG.B.2 to sub-cluster ‘A’ would result in slowing down the job. Similarly, the IS.B.4.9940 was not migrated to sub-cluster ‘B’ because its potential time gain was below the threshold. We feel that, *out-of-band* heuristics can be added to further optimize ARRIVE-F. It may also be noted that FT.B.8.1093 was not swapped with CG.B.2.1647, as the minimum threshold requirement was not met.

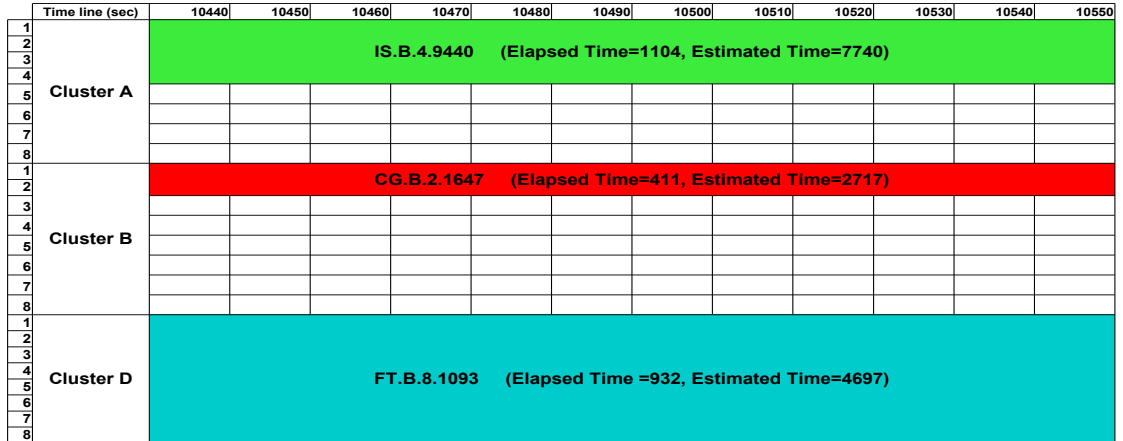


Figure 10: ARRIVE-F not migrating a 2 process job to make way for 8 process job.

Another interesting observation is a case of cyclic migrations (ping-pong migrations). In this case, a job may keep migrating between two sub-clusters. This only occurs in a case where the two sub-clusters having almost similar hardware

Table 21: ARRIVE-F not migrating a 2 process job... [Job Queue]

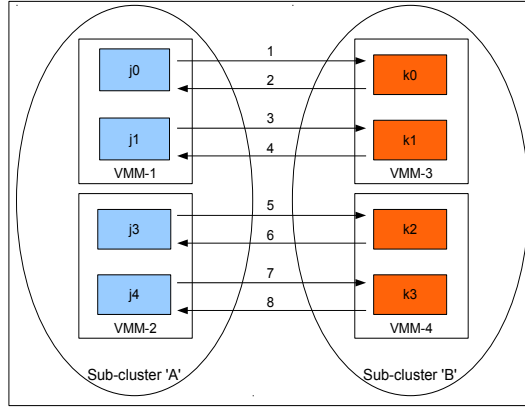
Jobs in the Scheduler Queue		
Name	No. of Processes	Estimated Time
LU.B.8.5	8	4
LU.B.8.16	8	14
CG.A.8.60	8	12
FT.A.8.16	8	18

are allocated similar jobs. For example, if LU.B.8.X and LU.B.8.Y dispatched to sub-cluster ‘A’ and sub-cluster ‘B’ respectively, where X and Y represent the number of iterations. For an aggressive value of T^{Thresh} ($0.05\beta\tau\eta$) we have observed cyclic migrations between the jobs, i.e. after every $\beta\tau$ seconds, the jobs swap their respective sub-clusters. This case is specific to LU.B.8 benchmark on our HC for a low threshold value, as it has relatively lower prediction accuracy. The cyclic migrations can affect the throughput of the compute farm in a negative way. As both the sub-clusters will remain busy for a significant time. One can add a simple heuristic to stop such a scenario by limiting the number of migrations of a particular job to and from a particular sub-cluster. Note that this case did not occur with our experiments detailed in Section 8, as the value of threshold was higher.

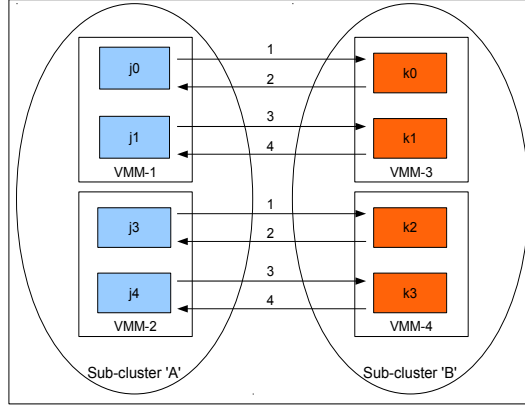
The profiling overheads of PMPI in ARRIVE-F are around 3%. This can be further reduced by dynamically attaching and de-attaching the pmpi library to an application. If the estimated time of an application is greater than the minimum time an application should run to qualify for migration, then at the initialization phase of the application PMPI library can be attached to generate the communication profile. As ARRIVE-F inherently suits iterative applications, subsequent profiles will be similar. The PMPI library can then be de-attached after a certain time period blocks. The same can be done with the CPU profiling. This will result in reducing the overheads of profiling substantially.

Currently, the migration assistant performs the migration in serial, i.e. at any given time only one node is migrated in the whole compute farm. The migration assistant can be parallelized so that it can perform multiple migrations simultaneously.

For example, the migration between two jobs j and k that span across multiple VMMs on sub-clusters ‘A’ and ‘B’ respectively. In the current form, the migration is carried out serially as shown in Figure 13(a). However, if the migration is done in parallel as shown in Figure 13(b), the migration time will be greatly reduced. This will have a profound affect the wall clock time stretch of the a applications spanning multiple VMMs, in case of a large HC. To validate this assertion, we have conducted a set of experiments on a 2×2 sub-cluster ‘D’, as shown in Figure 12. The x-axis shows the benchmark and the y-axis gives the wall clock time stretch. We have used the live migration optimization modification for this experiment as with all the previous experiments. The number of iterations of benchmarks were changed as their original wall clock times were not sufficient to



(a) Serial migration of jobs spanning multiple VMMs.



(b) Parallel migration of Jobs spanning multiple VMMs.

Figure 11: Serial vs parallel migration of two jobs spanning multiple VMMs

conduct the migration runs. The experiment clearly suggests that the parallel migration will bring the wall clock time stretch down. We believe the wall clock time stretch will remain almost constant in case of parallel migration.

The serial and parallel migration did not affect the migration model in a small HC. As seen from the experiments, the cost of migration T^m is only a fraction of the potential time saved by the compute cluster. However, implementing this will result in an increased throughput for ARRIVE-F. In the next version of ARRIVE-F, we plan to introduce parallel migration.

In a related issue, shown in Figure 13, when migrating job ‘j’ and ‘k’, the migration assistant should also be able to do migration of jobs ‘L’ and ‘m’, if these are on a different set of VMMs or sub-clusters. The serial implementation in the current version of ARRIVE-F puts the job migration of the later on hold even if these jobs can be migrated without any loss of time.

However, for a small compute farm like the one used in the experimental setup, the serial implementation does not affect the throughput as reflected in

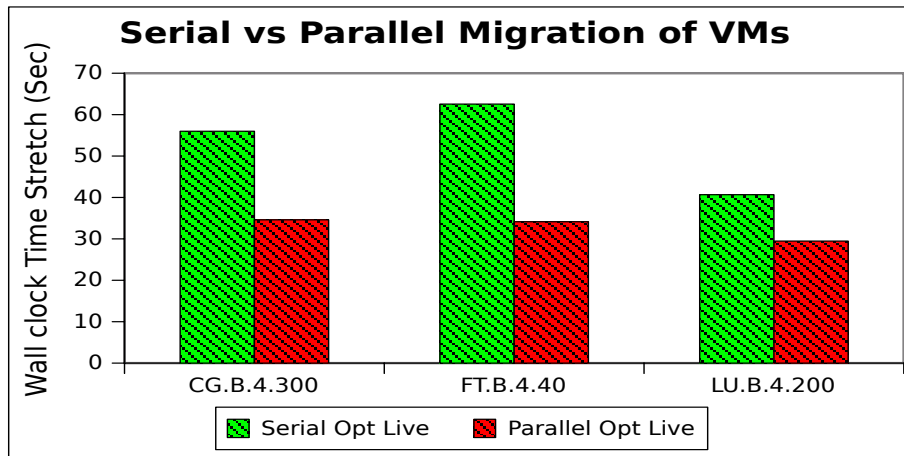
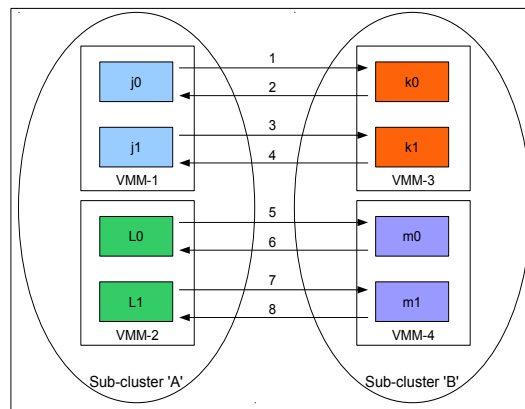
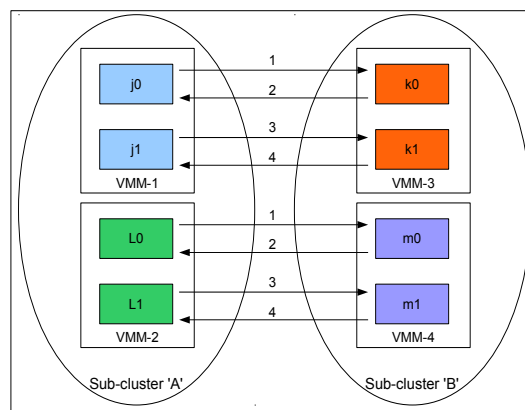


Figure 12: Impact of parallel migration.



(a) Serial migration of jobs spanning multiple VMMs.



(b) Parallel migration of Jobs spanning multiple VMMs.

Figure 13: Serial vs parallel migration of multiple jobs spanning multiple VMMs

Experiment 1. During the migration of FT.B.4.20 and FT.A.4.156, the jobs MG.B.8.5132 and FT.B.8.506 could have been migrated, but serial implementation prevented that. This only resulted in less than 200 seconds delay for the migration of the later to commence. The potential time lost was roughly less than 60 seconds, if the migration had started 200 seconds earlier. However, it may have a greater impact on an HC with high number of migrations can take place simultaneously.

The average waiting time and the turnaround times in our experiments are not in proportion to the throughput improvement and the overall time saved. Especially for Experiment 3, the result has a very high variation. The throughput improvement is 13% compared to average waiting time of went down by almost 300%. This is due to the nature of the workload queue given to the experiment. The queue consisted of 194 jobs. The longer running jobs held the jobs in the scheduler queue for a long time in the base-run. The jobs late in the queue, therefore could not be backfilled. This resulted in high average waiting time for the whole cluster. Towards the end of experimental run, most of the shorter jobs executed on the fastest sub-cluster ('B') making up the time. This resulted in relatively higher than expected average waiting time for the HC compared to the throughput improvement.

10 Appendix A

This appendix contains the list of jobs provided to ARRIVE-F for throughput experiments in Section 8. The job stream was generated using Lublin-Feitelson method and details are provided in same section.

10.1 Job Data - Experiment 1

The list of jobs provided to the scheduler for Experiment 1 in Section 8.1

Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
1	0	cg.A.4.452	4	79
2	0	lu.A.1.25	1	17
3	2	is.B.4.29	4	23
4	3	mg.B.4.14	4	16
5	4	mg.B.8.5132	8	2697
6	5	ft.A.4.750	4	95
7	21	cg.B.2.27	2	44
8	26	is.A.8.204	8	52
9	38	ft.A.2.39	2	71
10	43	mg.B.1.39	1	21
11	46	cg.A.4.104	4	27
12	51	ep.B.6.0	6	25
13	55	ft.A.4.6	4	8
14	58	cg.A.8.560	8	112
15	60	ep.B.1.0	1	150
16	61	ft.A.4.156	4	230
17	67	ep.A.8.0	8	7
18	72	ep.B.1.0	1	150
19	74	cg.A.2.10	2	2
20	83	ep.A.8.0	8	7
21	89	ft.A.8.13	8	14
22	101	cg.A.2.764	2	168
23	104	ft.B.8.506	8	2174
24	107	lu.B.2.53	2	108
25	109	ep.A.3.0	3	16
26	110	ep.B.5.0	5	6
27	113	is.A.8.10655	8	2717
28	115	mg.A.4.4	4	4
29	116	ep.A.5.0	5	14
30	122	ft.B.8.3	8	12
31	125	lu.A.8.13367	8	2005
32	130	lu.B.4.10	4	18
33	131	ep.B.3.0	3	12
34	132	lu.B.8.288	8	253
35	137	lu.B.4.102	4	191
36	147	ep.B.6.0	6	28
37	148	ep.A.5.0	5	11
38	150	ep.A.5.0	5	11
39	150	ep.B.4.2	4	40
40	157	ep.A.5.0	5	12
41	160	ep.B.3.1	3	5
42	160	lu.B.2.3	2	6
43	167	is.A.8.499	8	127
44	168	is.A.4.2014	4	450
45	169	ft.A.4.7	4	10
46	172	ep.A.6.0	6	10

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
47	174	lu.B.8.141	8	124
48	183	cg.A.4.27	4	7
49	200	ep.A.7.0	7	7
50	215	ft.A.2.2	2	3
51	218	ep.B.5.0	5	9
52	222	ep.B.5.0	5	29
53	223	lu.A.8.67	8	10
54	231	cg.B.4.2	4	2
55	231	mg.B.2.10	2	11
56	241	mg.A.4.6	4	6
57	245	ep.B.7.0	7	14
58	249	lu.B.4.14	4	25
59	251	lu.B.1.50	1	164
60	251	mg.A.4.11	4	12
61	253	is.B.4.19	4	15
62	254	lu.A.2.942	2	452
63	257	mg.A.2.31	2	34
64	260	cg.B.4.4	4	5
65	260	ft.A.2.17	2	31
66	265	ft.A.8.291	8	328
67	273	lu.A.4.97	4	26
68	274	ft.A.4.2	4	2
69	277	cg.B.8.267	8	290
70	287	ep.A.8.11	8	7
71	288	ep.B.2.4	2	70
72	298	lu.A.4.52	4	14
73	300	ep.A.8.32	8	8
74	300	lu.A.1.318	1	222
75	308	mg.A.4.145	4	170
76	322	is.A.8.55	8	14
77	328	ep.A.2.1	2	25
78	355	ep.B.6.0	6	25
79	369	lu.B.1.34	1	111
80	371	ft.A.4.142	4	210
81	389	ep.B.5.0	5	33
82	393	ep.B.3.0	3	2
83	414	lu.A.1.39	1	27
84	417	ft.A.2.440	2	813
85	424	ft.A.8.40	8	45
86	444	mg.B.2.20	2	22
87	455	mg.A.4.4	4	4
88	463	mg.B.2.47	2	52
89	466	ep.A.4.1	4	16
90	468	mg.A.2.47	2	52
91	471	is.A.4.59	4	13
92	473	ep.B.5.0	5	38
93	476	ep.A.5.0	5	13
94	480	lu.A.8.134	8	20
95	484	is.A.8.8	8	2
96	487	cg.A.4.166	4	43
97	487	ft.A.4.5	4	6
98	494	mg.A.4.25	4	29
99	497	ep.A.5.0	5	10
100	516	ft.A.4.40	4	58
101	528	is.B.8.463	8	354
102	535	cg.B.4.2286	4	3268
103	543	lu.A.1.7385	1	5169
104	549	cg.A.8.1420	8	284

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
105	555	ft.A.4.53	4	78
106	567	ep.A.8.37	8	7
107	570	ft.B.8.462	8	1984
108	575	lu.A.8.12334	8	1850
109	578	ep.A.3.1	3	16
110	590	lu.B.1.455	1	1500
111	619	ep.B.5.0	5	39
112	638	mg.B.2.113	2	127
113	676	is.A.4.314	4	70
114	680	mg.A.4.44	4	51
115	701	cg.B.4.11	4	15
116	701	ep.B.5.0	5	36
117	715	cg.A.4.66	4	17
118	724	lu.A.4.304	4	82
119	743	cg.B.8.57	8	62
120	750	cg.A.2.223	2	49
121	764	mg.B.8.6	8	8
122	782	ep.A.2.3	2	25
123	801	mg.A.4.12	4	14
124	805	ft.A.4.207	4	305
125	810	ft.A.4.245	4	362
126	815	mg.B.8.2	8	2
127	823	ep.B.7.0	7	16
128	825	cg.A.4.24	4	6
129	834	cg.B.8.254	8	276
130	840	mg.B.8.157	8	235
131	848	cg.A.8.550	8	110
132	861	cg.A.4.150	4	39
133	863	ep.B.5.0	5	33
134	867	ep.A.3.1	3	15
135	885	cg.A.2.46	2	10
136	888	mg.A.2.1	2	1
137	894	is.B.2.87	2	77
138	924	ft.A.4.2	4	2
139	961	lu.A.1.10	1	7
140	965	lu.B.4.731	4	1374
141	971	ep.A.8.3	8	9
142	975	lu.A.4.15	4	4
143	991	ep.B.2.1	2	73
144	994	ep.A.6.10	6	8
145	1021	is.B.4.7	4	5
146	1029	ep.A.5.0	5	13
147	1048	ep.A.5.0	5	12
148	1057	is.A.8.436	8	111
149	1060	lu.A.8.474	8	71
150	1068	lu.B.1.202	1	664
151	1068	ep.B.4.2	4	37
152	1079	ep.A.1.0	1	42
153	1082	ft.B.8.5	8	20
154	1100	ep.B.3.0	3	50
155	1103	cg.A.2.41	2	9
156	1109	mg.B.4.1	4	1
157	1114	cg.A.4.74	4	19
158	1126	ep.B.2.1	2	70
159	1139	is.A.4.103	4	23
160	1144	ep.A.3.2	3	16
161	1151	ep.B.5.0	5	35
162	1172	mg.B.8.2	8	3

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
163	1198	cg.A.8.295	8	59
164	1222	mg.B.8.1040	8	1560
165	1232	lu.B.1.137	1	452
166	1251	mg.A.4.28	4	33
167	1261	cg.A.4.174	4	45
168	1292	mg.A.2.150	2	168
169	1312	ep.A.1.0	1	54
170	1379	mg.A.2.267	2	298
171	1465	ep.A.1.0	1	39
172	1475	cg.A.8.530	8	106
173	1478	ep.B.8.11	8	19
174	1494	ft.A.4.413	4	611
175	1578	lu.B.1.10	1	33
176	1641	lu.A.1.86	1	60
177	1648	cg.A.8.15	8	3
178	1667	ft.B.8.8	8	33
179	1678	mg.B.4.75	4	89
180	1691	ft.A.4.3	4	4
181	1705	mg.B.4.10	4	11
182	1719	is.A.4.282	4	63
183	1722	ft.A.4.8	4	11
184	1723	cg.B.8.15	8	16
185	1730	lu.A.4.52	4	14
186	1738	cg.B.2.1	2	1
187	1740	ep.A.4.11	4	17
188	1746	mg.A.4.3	4	3
189	1755	is.A.4.81	4	18
190	1758	ft.A.8.925	8	1045
191	1759	cg.A.2.100	2	22
192	1761	cg.B.4.20	4	28
193	1811	mg.A.4.41	4	48
194	1815	lu.A.4.26	4	7
195	1840	lu.A.4.15	4	4
196	1847	cg.B.4.57	4	81
197	1850	lu.B.4.3	4	4
198	1853	ep.A.6.10	6	7
199	2129	ep.A.2.2	2	25
200	2170	ep.A.3.1	3	14
201	2177	lu.B.2.5	2	10
202	2184	lu.A.2.5	2	2
203	2184	lu.A.2.5	2	2
204	2218	ep.A.2.6	2	25
205	2228	mg.B.2.26	2	29
206	2229	lu.B.8.13	8	11
207	2233	cg.A.2.32	2	7
208	2243	lu.A.4.15	4	4
209	2263	cg.B.2.19	2	30
210	2267	ft.B.8.102	8	436
211	2270	lu.A.8.7	8	1
212	2271	lu.B.4.4	4	7
213	2274	ep.B.1.0	1	158
214	2276	ep.A.3.0	3	11
215	2277	lu.A.1.20	1	14
216	2279	lu.A.4.12	4	3
217	2280	mg.A.8.158	8	85
218	2291	ep.A.3.0	3	15
219	2293	cg.A.4.1877	4	488
220	2297	mg.B.4.6	4	6

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
221	2300	ep.B.4.1	4	38
222	2301	cg.B.8.0	8	31
223	2304	lu.B.1.0	1	611
224	2307	cg.A.2.328	2	72
225	2308	ft.A.4.24	4	35
226	2315	cg.B.4.607	4	868
227	2317	ep.B.7.0	7	17
228	2323	lu.A.4.30	4	14
229	2324	ft.B.8.83	8	355
230	2324	mg.A.1.70	1	297
231	2348	lu.B.8.10	8	17
232	2349	ep.A.6.0	6	8
233	2352	ep.A.6.0	6	7
234	2356	mg.A.8.4	8	4
235	2357	lu.B.8.19	8	16
236	2361	cg.A.8.1405	8	281
237	2367	lu.A.8.9774	8	1466
238	2371	lu.A.1.55	1	38
239	2381	cg.B.4.7	4	9
240	2386	ep.A.3.1	3	18
241	2390	ep.A.8.0	8	6
242	2396	ep.A.6.0	6	9
243	2430	lu.B.1.4	1	10
244	2436	ep.A.7.0	7	8
245	2438	lu.B.8.13	8	11
246	2460	cg.B.4.667	4	953
247	2464	cg.A.4.31	4	8
248	2466	is.A.4.18	4	4
249	2486	ft.A.4.2	4	2
250	2503	ep.A.4.4	4	18
251	2506	ft.A.4.3	4	3
252	2533	lu.B.8.427	8	375
253	2546	ft.B.8.23	8	95
254	2549	ep.A.4.1	4	14
255	2555	is.A.2.18	2	4
256	2563	mg.B.8.8	8	12
257	2570	ep.B.3.0	3	57
258	2570	is.A.8.428	8	109
259	2574	mg.B.2.424	2	478
260	2578	mg.A.4.11	4	12
261	2585	lu.B.8.612	8	538
262	2622	ft.A.4.3	4	4
263	2628	ft.A.4.1	4	1
264	2636	ft.A.8.197	8	222
265	2636	mg.B.4.6	4	6
266	2647	is.B.8.17	8	13
267	2676	cg.A.8.60	8	12
268	2693	lu.A.1.16	1	11
269	2711	lu.B.1.10	1	32
270	2716	lu.A.2.2048	2	983
271	2726	lu.A.8.720	8	108
272	2734	lu.A.4.8	4	2
273	2744	ft.A.4.11	4	16
274	2747	mg.B.4.9	4	10
275	2759	ft.A.4.2	4	2
276	2776	ft.A.4.8	4	11
277	2789	mg.A.4.6	4	7
278	2799	cg.B.8.138	8	150

continued on next page

continued from previous page

Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
279	2806	lu.A.1.92	1	64
280	2806	cg.B.8.412	8	448
281	2808	cg.B.8.41	8	44
282	2811	ft.A.8.39	8	43
283	2814	cg.A.4.43	4	11
284	2817	cg.A.4.420	4	109
285	2844	ft.B.8.3	8	9
286	2851	cg.A.4.208	4	54
287	2885	cg.B.8.69	8	75
288	2900	ep.A.7.0	7	7
289	2901	lu.B.4.4210	4	7914
290	2905	ft.A.4.171	4	252
291	2910	mg.B.4.958	4	1139
292	2911	mg.A.8.62	8	33
293	2931	ft.A.2.92	2	170
294	2933	mg.A.8.6	8	3
295	2955	lu.B.4.2	4	2
296	2956	ep.B.4.1	4	40
297	2961	lu.B.4.2	4	3
298	2972	ep.B.2.1	2	74
299	2973	ep.A.4.1	4	14
300	2973	lu.B.4.621	4	1167
301	2977	ep.A.3.1	3	15
302	2980	cg.B.8.29	8	31
303	2984	ft.A.2.5	2	9
304	2986	ft.A.2.20	2	37
305	2989	lu.B.2.3	2	6
306	2998	mg.B.4.6	4	7
307	2998	ft.B.8.5	8	19
308	3002	ft.A.8.7	8	7
309	3009	cg.B.8.9	8	9
310	3012	ft.A.4.3	4	3
311	3031	mg.B.2.35	2	39
312	3038	ep.B.8.5	8	19
313	3043	ft.B.8.206	8	883
314	3049	mg.A.4.74	4	87
315	3054	lu.A.2.7	2	3
316	3062	ft.A.2.1003	2	1854
317	3063	ep.A.8.2	8	8
318	3084	ft.A.4.93	4	137
319	3087	mg.A.4.325	4	383
320	3116	ep.B.2.1	2	73
321	3120	lu.B.8.121	8	106
322	3125	mg.A.4.4	4	4
323	3131	ft.A.8.435	8	491
324	3146	ep.A.4.22	4	18
325	3149	cg.A.4.3897	4	1013
326	3154	lu.B.4.130	4	243
327	3157	cg.A.8.410	8	82
328	3161	lu.B.4.4	4	7
329	3170	lu.A.4.19	4	5
330	3173	ep.A.4.161	4	17

Table 22: Experiment 1: List of jobs

10.2 Job Data - Experiment Number 2

The list of jobs provided to the scheduler for Experiment 2 in Section 8.2.

Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
1	0	mg.B.2.15	2	16
2	1	lu.A.8.527	8	79
3	1	is.B.8.40	8	30
4	4	is.B.8.23	8	17
5	7	lu.A.8.51600	8	7740
6	13	is.B.2.14	2	12
7	16	lu.B.4.5	4	8
8	18	mg.B.8.8	8	5
9	19	ep.B.4.1	4	36
10	23	cg.B.4.2	4	2
11	30	mg.A.2.16	2	17
12	37	cg.A.8.290	8	58
13	38	ep.A.8.2	8	8
14	39	lu.A.1.390	1	273
15	44	ft.A.8.1	8	1
16	49	mg.B.4.5	4	5
17	52	lu.B.4.1039	4	1952
18	54	mg.B.4.83	4	98
19	59	mg.B.8.245	8	132
20	63	ep.A.8.1	8	14
21	65	lu.A.1.48	1	33
22	68	lu.B.1.2	1	13
23	71	ft.A.4.32	4	46
24	71	ep.A.4.2	4	15
25	75	mg.A.1.4	1	17
26	76	ft.A.4.16	4	23
27	79	ep.A.8.2	8	15
28	83	lu.A.4.7426	4	2005
29	85	cg.B.8.22	8	23
30	102	mg.B.8.30	8	16
31	110	is.A.4.77	4	17
32	122	lu.A.4.193	4	52
33	134	cg.A.4.274	4	71
34	141	ep.B.2.1	2	72
35	146	cg.A.2.123	2	27
36	170	lu.B.2.41	2	83
37	177	ft.A.8.8	8	8
38	180	lu.A.4.415	4	112
39	183	mg.B.4.61	4	72
40	184	lu.B.4.123	4	230
41	190	lu.A.1.118	1	82
42	192	mg.B.8.4	8	2
43	202	is.B.4.10	4	8
44	208	cg.A.8.70	8	14
45	222	lu.A.8.1120	8	168
46	227	ep.A.4.262	4	18
47	231	ep.A.4.14	4	15
48	234	is.A.4.50	4	11
49	236	mg.A.4.6	4	6
50	239	mg.A.8.5032	8	2717
51	242	lu.A.4.15	4	4
52	243	ep.A.4.2	4	14
53	250	cg.A.8.60	8	12
54	253	mg.B.8.56	8	30

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
55	256	ft.B.8.1093	8	4697
56	262	ep.B.4.1	4	38
57	266	ep.A.4.2	4	12
58	268	ep.A.8.61	8	15
59	274	is.B.4.233	4	191
60	285	ft.A.4.6	4	8
61	287	is.B.8.28	8	21
62	290	ep.B.2.1	2	67
63	291	lu.A.8.267	8	40
64	301	cg.B.4.2	4	2
65	304	is.A.4.23	4	5
66	305	is.A.4.27	4	6
67	315	cg.A.2.578	2	127
68	316	lu.B.8.512	8	450
69	318	cg.A.4.39	4	10
70	323	is.B.4.5	4	4
71	326	is.B.4.152	4	124
72	337	mg.A.4.6	4	7
73	355	is.B.8.31	8	23
74	372	lu.A.2.7	2	3
75	378	ft.A.8.8	8	9
76	383	cg.A.2.132	2	29
77	385	ep.B.1.0	1	150
78	395	ep.A.1.0	1	32
79	397	ft.A.8.10	8	11
80	407	ep.A.1.0	1	36
81	413	mg.B.1.26	1	14
82	418	ft.B.8.6	8	25
83	421	lu.B.8.187	8	164
84	422	mg.B.4.11	4	12
85	427	ep.B.4.1	4	45
86	428	cg.A.4.1739	4	452
87	432	lu.A.4.126	4	34
88	435	ft.A.4.4	4	5
89	435	mg.A.2.28	2	31
90	441	is.B.4.400	4	328
91	450	is.A.4.117	4	26
92	452	cg.B.8.2	8	2
93	455	lu.B.4.155	4	290
94	467	cg.A.4.162	4	42
95	473	cg.A.4.820	4	213
96	483	is.A.2.63	2	14
97	486	lu.A.8.887	8	133
98	487	ep.A.8.54	8	12
99	495	ep.A.4.21	4	17
100	509	ft.A.4.3	4	14
101	516	cg.B.8.14	8	15
102	544	ep.A.4.3	4	17
103	560	lu.A.4.412	4	111
104	562	is.B.8.275	8	210
105	581	is.A.2.276	2	62
106	587	lu.A.1.3	1	2
107	609	is.B.4.33	4	27
108	613	ft.A.8.720	8	813
109	620	ep.B.4.2	4	45
110	640	ft.B.8.6	8	22
111	653	lu.B.8.5	8	4
112	662	ep.A.1.0	1	35

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
113	667	ft.B.8.1	8	2
114	671	ep.A.8.13	8	15
115	675	ft.A.4.9	4	13
116	679	lu.B.1.54	1	78
117	682	ep.B.8.1	8	23
118	688	lu.A.8.134	8	20
119	693	mg.A.4.2	4	2
120	697	mg.A.2.39	2	43
121	699	ft.A.4.5	4	6
122	709	cg.B.4.21	4	29
123	713	lu.A.8.334	8	50
124	733	ep.B.8.4	8	25
125	746	is.A.4.1584	4	354
126	753	lu.B.1.810	1	3268
127	760	is.B.4.347	4	284
128	767	is.A.8.306	8	78
129	781	is.A.8.597	8	152
130	786	is.A.8.7781	8	1984
131	791	lu.A.8.12334	8	1850
132	796	cg.A.8.20	8	4
133	808	is.A.4.6712	4	1500
134	839	lu.B.4.11	4	19
135	860	ft.A.8.113	8	127
136	898	lu.A.8.467	8	70
137	902	cg.A.4.197	4	51
138	925	is.A.4.68	4	15
139	926	is.B.2.18	2	16
140	940	ft.A.2.10	2	17
141	950	cg.B.8.76	8	82
142	969	mg.B.2.55	2	62
143	977	ep.A.4.6	4	14
144	992	ft.A.4.6	4	8
145	1010	mg.B.4.33	4	39
146	1030	ep.B.4.1	4	44
147	1034	cg.A.2.1387	2	305
148	1040	ft.A.4.245	4	362
149	1047	ep.B.8.1	8	20
150	1058	mg.A.4.14	4	16
151	1062	ft.A.4.5	4	6
152	1071	lu.A.4.1023	4	276
153	1078	lu.B.1.300	1	1295
154	1088	cg.A.4.424	4	110
155	1103	is.A.8.153	8	39
156	1106	ft.A.4.87	4	128
157	1111	ep.A.8.2	8	15
158	1130	lu.A.1.16	1	10
159	1133	lu.A.1.2	1	1
160	1142	cg.B.4.54	4	77
161	1173	ft.A.8.2	8	2
162	1210	ft.A.8.7	8	7
163	1217	cg.B.8.1261	8	1374
164	1223	is.B.8.12	8	9
165	1228	ep.B.4.1	4	46
166	1246	ft.A.4.9	4	13
167	1250	mg.B.4.2	4	2
168	1280	cg.A.4.20	4	5
169	1290	mg.B.4.3	4	3
170	1379	is.A.4.54	4	12

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
171	1418	is.A.8.436	8	111
172	1424	ft.A.4.48	4	71
173	1434	ep.A.4.80	4	14
174	1436	lu.A.8.247	8	37
175	1448	mg.A.8.41	8	22
176	1452	mg.A.4.17	4	20
177	1470	mg.A.8.2	8	1
178	1473	cg.B.4.7	4	9
179	1481	ep.A.8.1	8	10
180	1487	cg.A.4.74	4	19
181	1502	mg.B.8.15	8	8
182	1518	lu.A.1.33	1	23
183	1523	ft.A.4.3	4	11
184	1531	mg.B.4.16	4	19
185	1553	ep.B.8.1	8	14
186	1817	ft.A.8.53	8	59
187	1841	lu.A.1.2229	1	1560
188	1853	lu.A.8.3014	8	452
189	1872	lu.A.2.69	2	33
190	1885	ep.A.2.3	2	20
191	1918	mg.A.2.150	2	168
192	1939	ep.A.4.7	4	19
193	2008	lu.A.4.1104	4	298
194	2094	mg.A.4.299	4	352
195	2105	ft.B.8.25	8	106
196	2108	cg.B.4.119	4	170
197	2126	ft.A.4.413	4	611
198	2211	ft.A.4.23	4	33
199	2275	mg.B.4.51	4	60
200	2282	ep.A.2.1	2	23
201	2301	mg.B.4.28	4	33
202	2315	lu.B.8.102	8	89
203	2329	lu.B.1.2	1	8
204	2345	mg.A.4.10	4	11
205	2361	cg.A.8.315	8	63
206	2364	is.B.8.15	8	11
207	2366	ft.A.4.11	4	16
208	2373	lu.B.4.8	4	14
209	2381	ft.B.8.1	8	1
210	2383	mg.B.8.162	8	87
211	2391	cg.A.2.14	2	3
212	2401	ft.A.8.16	8	18

Table 23: Experiment 2: List of jobs

10.3 Job Data - Experiment Number 3

The list of jobs provided to the scheduler for Experiment 3 in Section 8.1. This experiment does not contain the sub-cluster ‘C’ which uses 100 mbps network.

Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
1	0	cg.B.4.12	4	16
2	0	ep.B.4.3	4	79
3	0	cg.A.4.116	4	30
4	52	cg.A.4.0	4	17
5	55	mg.A.8.23	8	12
6	94	lu.B.8.5	8	4
7	110	mg.B.4.7	4	8
8	158	lu.A.4.19	4	5
9	314	ft.A.4.2	4	2
10	376	ep.A.2.1	2	2
11	394	ft.A.4.12	4	17
12	422	ep.A.4.7	4	58
13	590	lu.A.1.1	1	8
14	599	ep.A.4.33	4	273
15	760	ft.A.4.7	4	9
16	785	is.A.2.23	2	5
17	814	cg.B.2.1184	2	1952
18	857	ep.B.2.2	2	98
19	909	lu.B.8.150	8	132
20	926	lu.B.2.2	2	4
21	1024	lu.A.8.220	8	33
22	1061	mg.A.4.2	4	3
23	1070	ep.B.4.2	4	46
24	1149	ep.A.8.4	8	15
25	1168	ep.B.2.1	2	17
26	1182	is.A.4.103	4	23
27	1187	cg.B.8.5	8	5
28	1220	lu.B.8.2279	8	2005
29	1227	lu.A.8.154	8	23
30	1289	is.A.4.72	4	16
31	1298	cg.B.4.3	4	44
32	1326	mg.A.4.45	4	52
33	1362	mg.A.4.61	4	71
34	1368	cg.B.4.15	4	21
35	1403	lu.A.4.100	4	27
36	1417	lu.B.2.41	2	83
37	1428	cg.A.4.31	4	8
38	1480	cg.B.2.68	2	112
39	1518	is.A.8.283	8	72
40	1635	mg.A.8.4	8	230
41	1641	mg.B.8.152	8	82
42	1653	is.B.4.9440	4	7740
43	1684	ft.A.4.2	4	2
44	1688	ep.A.4.1	4	8
45	1771	mg.A.8.26	8	14
46	1801	ep.B.1.0	1	168
47	1835	lu.B.4.1157	4	2174
48	1845	lu.B.4.58	4	108
49	1856	lu.A.2.5	2	11
50	1880	is.A.8.24	8	6
51	1919	ft.B.8.1093	8	4697
52	1925	lu.B.8.5	8	4
53	1941	lu.B.8.16	8	14

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
54	1980	cg.A.8.60	8	12
55	1982	cg.B.2.1647	2	2717
56	1989	ft.A.8.16	8	18
57	2024	lu.B.4.7	4	12
58	2068	mg.A.2.6	2	253
59	2202	cg.A.4.735	4	191
60	2307	cg.A.4.31	4	8
61	2390	ep.A.2.2	2	21
62	2407	mg.A.2.16	2	17
63	2477	ep.A.4.5	4	40
64	2506	mg.B.2.2	2	2
65	2632	mg.B.2.5	2	5
66	2651	cg.A.2.28	2	6
67	2662	cg.A.4.7	4	127
68	2676	ft.A.8.399	8	450
69	2677	cg.B.4.7	4	10
70	2687	cg.A.8.20	8	4
71	2702	mg.B.2.110	2	124
72	2714	mg.A.4.6	4	7
73	2722	mg.A.2.21	2	23
74	2742	mg.B.2.3	2	3
75	2757	cg.A.2.41	2	9
76	2804	mg.B.8.8	8	29
77	2842	cg.A.4.39	4	10
78	2873	cg.A.2.10	2	2
79	2907	mg.B.4.10	4	11
80	2921	cg.A.2.28	2	6
81	2963	cg.B.8.13	8	14
82	3012	ep.A.8.6	8	25
83	3100	cg.B.4.115	4	164
84	3183	is.B.8.16	8	12
85	3314	mg.A.8.9	8	15
86	3329	mg.A.2.404	2	452
87	3343	ft.A.4.23	4	34
88	3357	cg.A.8.25	8	5
89	3461	cg.B.8.29	8	31
90	3466	mg.B.4.276	4	328
91	3479	mg.B.2.24	2	26
92	3527	lu.B.1.14	1	2
93	3576	cg.A.2.1319	2	290
94	3604	ep.A.4.10	4	42
95	3707	ep.B.4.7	4	213
96	3735	cg.A.2.64	2	14
97	3802	cg.A.4.512	4	133
98	3924	ep.B.2.4	2	222
99	3954	ep.B.4.6	4	170
100	4026	cg.A.8.70	8	14
101	4058	ep.B.4.1	4	15
102	4088	mg.A.8.32	8	17
103	4099	cg.B.4.11	4	111
104	4117	ft.A.4.142	4	210
105	4141	lu.A.4.230	4	62
106	4142	mg.B.8.4	8	2
107	4156	ft.A.2.15	2	27
108	4241	lu.A.4.3012	4	813
109	4250	mg.A.2.41	2	45
110	4337	ft.A.4.15	4	22
111	4340	ep.A.2.1	2	4

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
112	4369	ft.A.2.12	2	52
113	4492	is.B.2.3	2	2
114	4498	lu.A.8.347	8	52
115	4510	ep.A.1.0	1	13
116	4516	ep.B.2.1	2	8
117	4545	lu.B.1.20	1	3
118	4555	is.B.8.27	8	20
119	4583	ft.A.2.2	2	2
120	4584	lu.A.2.90	2	43
121	4604	ft.A.2.13	2	6
122	4628	is.A.4.130	4	29
123	4702	ep.B.4.2	4	50
124	4832	cg.B.2.36	2	58
125	4845	lu.B.8.403	8	354
126	4899	is.A.2.14525	2	3268
127	4911	lu.B.4.2750	4	5169
128	4913	cg.A.2.1291	2	284
129	4968	ft.A.8.70	8	78
130	4994	ep.A.4.14	4	152
131	5023	lu.B.8.2255	8	1984
132	5043	is.B.8.2419	8	1850
133	5046	is.A.4.18	4	4
134	5077	mg.A.4.1272	4	1500
135	5173	lu.B.4.11	4	19
136	5258	ep.B.4.4	4	127
137	5297	lu.B.4.38	4	70
138	5365	mg.A.2.46	2	51
139	5384	mg.A.2.15	2	15
140	5392	lu.B.2.8	2	16
141	5401	lu.B.4.10	4	17
142	5482	ep.A.4.10	4	82
143	5525	ep.B.8.4	8	62
144	5581	ep.A.8.12	8	49
145	5591	lu.B.8.10	8	8
146	5605	lu.A.4.145	4	39
147	5653	lu.B.2.7	2	14
148	5710	ft.A.8.16	8	305
149	5753	mg.B.4.305	4	362
150	5828	lu.A.1.3	1	2
151	6272	mg.A.4.14	4	16
152	6286	ep.B.4.1	4	6
153	6329	ep.A.4.34	4	276
154	6355	mg.B.8.436	8	235
155	6369	ft.A.4.75	4	110
156	6527	lu.B.1.260	1	39
157	6596	lu.A.1.17	1	128
158	6625	is.A.4.23	4	5
159	6691	lu.B.1.67	1	10
160	6747	is.A.4.5	4	1
161	6815	lu.B.2.38	2	77
162	6839	cg.B.4.2	4	2
163	6851	mg.B.2.7	2	7
164	6858	is.B.4.1676	4	1374
165	6862	lu.B.8.11	8	9
166	6869	lu.B.2.18	2	4
167	6944	cg.A.8.65	8	13
168	7033	lu.A.8.1	8	2
169	7049	cg.B.4.4	4	5

continued on next page

continued from previous page				
Sr. No	Arrival Time	Job Name (Benchmark)	Total Processes	Expected Time
170	7066	mg.A.4.3	4	3
171	7210	mg.B.4.11	4	12
172	7223	is.A.8.436	8	111
173	7242	lu.B.4.38	4	71
174	7261	ft.A.4.449	4	664
175	7281	lu.B.8.19	8	37
176	7320	cg.A.4.85	4	22
177	7329	cg.B.4.14	4	20
178	7371	mg.A.2.1	2	1
179	7385	lu.A.1.13	1	9
180	7635	cg.B.2.1	2	1
181	7664	ep.B.8.2	8	19
182	7676	mg.B.4.7	4	8
183	7692	cg.A.4.89	4	23
184	7694	lu.A.1.20	1	11
185	7811	lu.B.8.22	8	19
186	7931	ep.A.8.1	8	3
187	7975	mg.A.2.53	2	59
188	8000	lu.B.4.241	4	452
189	8141	ep.B.8.2	8	33
190	8249	cg.A.4.174	4	45
191	8274	lu.A.8.1120	8	168
192	8296	ft.A.2.21	2	54
193	8470	cg.B.2.181	2	298
194	8496	mg.B.4.296	4	352

Table 24: Experiment 3: List of jobs

References

- [1] OpenMPI.
<http://www.open-mpi.org/>, June 2008.
- [2] AMD Corp.
<http://www.amd.com>, Oct 2010.
- [3] NAS Parallel Benchmarks.
<http://www.nas.nasa.gov/Software/NPB>, Sep 2010.
- [4] OSU Benchmarks.
<http://mvapich.cse.ohio-state.edu/benchmarks/>, Oct 2010.
- [5] ABURTO, A.
Flops c program (double precision) v2.0.
<ftp://ftp.nosc.mil/pub/aburto/flops> [original site, not accessible]
<http://cs.anu.edu.au/~Muhammad.Atif/opensource/flops.c> [copy]
<http://www.mathworks.com/matlabcentral/fileexchange/7996-cpu-floating-point-operations-per-second-for-windows> [copy], Dec 1992.
- [6] ARMSTRONG, R., HENSGEN, D., AND KIDD, T.
The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions.
Heterogeneous Computing Workshop 0 (1998), 79.

- [7] ATIF, M., AND STRAZDINS, P.
An evaluation of multiple communication interfaces in virtualized smp clusters.
In *HPCVirt '09: Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*. (2009), ACM, pp. 9–16.
- [8] ATIF, M., AND STRAZDINS, P.
Optimizing live migration of virtual machines in smp clusters for hpc applications.
Network and Parallel Computing Workshops, IFIP International Conference on 0 (2009), 51–58.
- [9] BAILEY, D. H., AND SNAVELY, A.
Performance modeling: understanding the present and predicting the future.
In *Europar* (2005).
- [10] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A.
Xen and the art of virtualization.
In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM, pp. 164–177.
- [11] BRAUN, T. D., SIEGEL, H. J., BECK, N., BOLONI, L. L., MAHESWARAN, M., REUTHER, A. I., ROBERTSON, J. P., THEYS, M. D., YAO, B., HENSGEN, D., AND FREUND, R. F.
A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems.
Journal of Parallel and Distributed Computing 61 (2001), 810–837.
- [12] DONGARRA, J., AND LASTOVETSKY, A.
An overview of heterogeneous high performance and grid computing.
In *Engineering the Grid: Status and Perspective* (2006), B. D. Martino, J. Dongarra, A. Hoisie, L. Yang, and H. Zima, Eds.
- [13] DOWNEY, A.
A parallel workload model and its implications for processor allocation.
pp. 112–123.
- [14] IBARRA, O. H., AND KIM, C. E.
Heuristic algorithms for scheduling independent tasks on nonidentical processors.
J. ACM 24, 2 (1977), 280–289.
- [15] INNOVATIVE COMPUTING LABORATORY, U. O. T.
High performance linpack benchmark.
<http://www.netlib.org/benchmark/hpl/>, 3 2009.
- [16] KATRAMATOS, D., AND CHAPLIN, S. J.

- A cost/benefit estimating service for mapping parallel applications on heterogeneous clusters.
In *IEEE International Conference on Cluster Computing (Cluster 2005)* (2005).
- [17] LUBLIN, U., AND FEITELSON, D. G.
The workload on parallel supercomputers: Modeling the characteristics of rigid jobs.
Journal of Parallel and Distributed Computing (November 2003), 1105–1122.
- [18] MENON, A., SANTOS, J. R., AND YOSHIO, T.
Diagnosing performance overheads in the xen virtual machine environment.
In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments* (2005), pp. 13–23.
- [19] MU'ALEM, A. W., AND FEITELSON, D. G.
Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling.
IEEE Trans. Parallel Distrib. Syst. 12 (June 2001), 529–543.
- [20] NAKAZAWA, M., LOWENTHAL, D. K., AND ZHOU, W.
The mheta execution model for heterogeneous clusters.
In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (Washington, DC, USA, 2005), IEEE Computer Society.
- [21] TSAFRIR, D., ETSION, Y., AND FEITELSON, D.
Modeling user runtime estimates.
In *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, Eds., vol. 3834 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 1–35.