Scheduling MapReduce jobs in HPC clusters *

Marcelo Veiga Neves, Tiago Ferreto, and César De Rose

Faculty of Informatics, PUCRS, Brazil
marcelo.neves@acad.pucrs.br, tiago.ferreto@pucrs.br,
cesar.derose@pucrs.br

Abstract. MapReduce (MR) has become a de facto standard for large-scale data analysis. Moreover, it has also attracted the attention of the HPC community due to its simplicity, efficiency and highly scalable parallel model. However, MR implementations present some issues that may complicate its execution in existing HPC clusters, specially concerning the job submission. While on MR there are no strict parameters required to submit a job, in a typical HPC cluster, users must specify the number of nodes and amount of time required to complete the job execution. This paper presents the MR Job Adaptor, a component to optimize the scheduling of MR jobs along with HPC jobs in an HPC cluster. Experiments performed using real-world HPC and MapReduce workloads have show that MR Job Adaptor can properly transform MR jobs to be scheduled in an HPC Cluster, minimizing the job turnaround time, and exploiting unused resources in the cluster.

1 Introduction

The MapReduce (MR) model is in increasing adoption by several researchers, including the ones that used to rely on HPC solutions [19, 18]. Much of this enthusiasm is due to the highly visible cases where MR has been successfully used by companies like Google, Yahoo, and Facebook. Besides, MR provides a simpler approach to address the parallelization problem over traditional approaches, such as MPI [10].

MR implementations, such as Hadoop [20], provide a complete execution platform for MR applications, normally using a dedicated cluster in combination with an optimized distributed file system. As consequence, in order to enable the execution of regular HPC and MR jobs in a computing laboratory, two distinct clusters are required. It leads to a split in the laboratory investments, in terms of hardware and staff, to support the two models, instead of focusing in a single, large scale and powerful computing infrastructure.

We believe that users and computing laboratory administrators may benefit from using already existing HPC clusters to execute MR jobs. In order to enable it, one of the first issues that must be addressed is regarding the job submission process. While MR implementations provide a straightforward job submission process which involves the whole cluster, HPC users submit their jobs to a

* The original publication is available at www.springerlink.com (Euro-Par 2012).

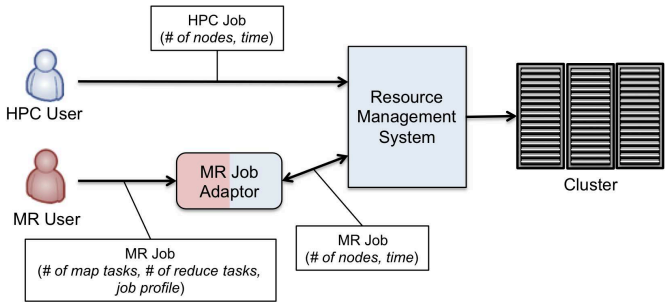

Fig. 1. Architecture of the HPC cluster with the MR Job Adaptor

Resource Management System (RMS) and need to specify the number of nodes and amount of time that should be allocated for complete the job execution.

Current solutions, such as Hadoop on Demand (HOD) [1] and myHadoop [13] allow one to create a virtual Hadoop cluster as a partition of a large physical cluster. However, the user must explicitly specify the number of nodes and time to be allocated as a regular HPC job. This approach may confuse typical MR users that are not used to do it, and they may, in return, always try to allocate the whole cluster for the longest time as possible. As a consequence, the turnaround time of the MR job will probably increase, since the job will be scheduled to the end of the RMS queue, which will frustrate the user again. Other solutions, such as MESOS [11] and Hamster [21], use a different approach where the responsibility for resource management is taken away from the cluster's RMS, which may conflict with the policy of use of most HPC clusters.

In order to overcome these problems, we present in this paper the MR Job adaptor, a component that converts MR jobs in order to enable their execution in HPC clusters. Figure 1 presents the MR Job adaptor and its connection with MR users and the RMS of the HPC Cluster. It receives the MR job from the user and interacts with the RMS in order to find a suitable slot to schedule the job that minimizes the resulting turnaround time of the job. We evaluated the algorithm implemented inside the MR Job Adaptor using real HPC and MR workloads and observed that it effectively decreases the turnaround time while also exploits unused resources in the cluster.

The paper is organized as follows: Section 2 provides an overview of HPC clusters and the Map Reduce model; Section 3 describes the functioning of the MR Job Adaptor for HPC Clusters; Section 4 presents the experiments performed to evaluate the MR Job Adaptor using real workloads. The conclusion and future work are presented in Section 5.

2 HPC clusters and MapReduce

Clusters of computers, which have transformed HPC in the last decade, are still the dominant architecture in this area [22]. HPC clusters consist of a number of stand-alone computers connected by a high performance network, working together as a single computing resource, and sharing a common storage volume exported through a distributed file system.

Traditional HPC clusters typically have their resources controlled by a Resource Management System (RMS), such as PBS/TORQUE [23] or SGE [16], which enable the submission, tracking and management of jobs in the cluster. Although this approach maximizes the overall utilization of the system and enables sharing of the resources among multiple users [13], it also requires all applications to be submitted as batch jobs. The user must submit the job accompanied by the number of nodes that the parallel application should use and the maximum time that it will take to complete the job execution.

The de-facto standard for parallel programming in HPC clusters is MPI (Message Passing Interface [10]), which follows the message-passing paradigm. A parallel program using MPI consists of different processes running on the cluster and explicitly exchanging data via messages. Despite the higher complexity in the development of parallel applications using this approach, it also enables one to fine-tune the application, resulting in better performance than other high-level approaches.

In the past few years, the increase in data generation has reached rates never seen before, making it necessary to develop new technologies for storing and analyzing such a large amount of data. The processing of such large data sets, also known as big data, is normally referred as data-intensive computing [15]. Several works have already been proposed in the data-intensive computing area to address the needs of big data [17, 6, 9]. In this model, the data set may not fit in the main memory nor in a single disk and, therefore, a distributed storage solution is necessary.

The execution platform for data-intensive computing is typically a dedicated large-scale cluster, with the data set distributed between the cluster nodes, i.e., each node has a slice of the data set. Thus, each node is both a data and compute node, which provides scalable storage and efficient data processing (by exploiting data locality). This architecture differs from traditional HPC clusters in some ways. HPC clusters are usually shared by several users to execute different applications, through the mediation of a RMS, while in data-intensive computing the cluster is usually dedicated to process large data sets of an unique organization. HPC clusters use a shared-disk file system to share data between nodes, while in data-intensive computing each cluster node uses its own local storage (shared-nothing). Consequently, users of data-intensive computing usually adopt a dedicated cluster for their applications.

There are several frameworks for the development of data-intensive applications [6, 7, 12], most of them based on the MapReduce (MR) model. Hadoop [20] is currently one of the most popular open-source MapReduce implementations. Unlike typical HPC parallel programming libraries, such as MPI, MR frameworks

hide much of the complexity of parallel programming from the programmer (for example, not requiring explicit data communications or application-specific logic to avoid communications). Current MR implementations allow automatic parallelization and distribution of computations on large clusters of commodity PCs, hiding the details of parallelization, fault tolerance, data distribution and load balancing [6].

The availability of several programming frameworks and the facilities to develop a parallel program has contributed to the adoption of MR by traditional HPC users. Instead of explicitly specifying the communication between processes and guaranteeing their coordination, the definition of simple map and reduce tasks seemed to be simpler in some cases. As a result, typical CPU-intensive HPC applications started being reimplemented using the MapReduce model (e.g. scientific application [19]) and, consequently, using a dedicated MR cluster.

Instead of using two clusters, one for HPC applications and another for MR applications, the RMS of the HPC cluster should also be able to schedule MR jobs. However, in order to do that, the MR job must include the number of nodes and amount of time to execute the job, which is not common in MR implementations. There are some initiatives in order to execute MR jobs in an HPC cluster. Systems such as Hadoop on Demand (HOD) [1] and myHadoop [13] allow one to create a virtual Hadoop cluster as a partition of a large physical cluster. Both systems use the TORQUE Resource Management System [23] to perform the allocation of nodes. However, the user has to specify the number of nodes and time to be allocated.

A straightforward solution would be to request the whole cluster for as long as possible to execute the job. Despite the simplicity of this approach, it can lead to longer turnaround times, since the request will probably go to the end of the RMS queue, which increases the time before the request is attended [5]. Due to the high flexibility of MR jobs (they can be executed with a variable number of nodes), the RMS could use a more intelligent approach, trying to fit the MR job in the free slots available in the RMS queue. Therefore, we propose a component called MR Job Adaptor, used to adjust the request to an HPC Cluster RMS, including number of nodes and amount of time, while ensuring that the turnaround time is minimized.

3 MapReduce Job Adaptor

This section presents the algorithm implemented inside the MapReduce Job Adaptor. The adaptor has three main goals: (i) to facilitate the execution of MR jobs in HPC clusters, (ii) to minimize the average turnaround time of MR jobs executed in an HPC cluster, and (iii) to exploit unused resources in the cluster resulted from the various shapes of HPC job requests.

MR job requests are quite different from HPC ones. They do not require any specific infrastructure parameter for submission, only straightforward application parameters such as number of map and reduce tasks. On the other hand,

HPC job requests require the number of nodes and amount of time to allocate a cluster partition. The approach used by systems such as Hadoop on Demand and myHadoop to run MR jobs in HPC clusters is to ask the user how many nodes and time should be allocated for a job. However, this approach is quite cumbersome since, in general, users do not have this kind of knowledge about their MR applications for different numbers of nodes and combinations of map and reduces tasks. In practice, this causes the user to allocate the maximum allowed amount of time and resources in the cluster, which may cause longer turnaround times and waste of resources.

The proposed adaptor aims to enable the transparent execution of MR jobs in the HPC cluster, i.e., the user specifies the MR job request as he would do in a typical MR cluster and the adaptor converts it to an HPC-compatible request, which is forwarded to the Resource Management System (RMS) of the HPC cluster. Instead of always using the maximum amount of nodes and time to execute the MR job, the adaptor allocates a cluster partition which minimizes the turnaround time of the job. It does that by interacting with the RMS to get free areas (slots) in the job requests queue. Using a profile of the MR job, it estimates the job completion time for each free slot and selects the one that yields the minimum turnaround time.

This approach relies on the fact that MR jobs do not have strict requirements regarding the number of resources for execution as HPC jobs. Thus, we use the MapReduce performance model proposed by Verma et al. [24] to estimate job completion times for different number of resources. It creates a job profile comprising performance invariants from past executions and uses it as input for the time estimation. This model can be used to estimate the lower (T_J^{low}) and upper (T_J^{up}) bounds of the overall completion time of a given job J. The lower bound can be obtained as follows:

$$T_J^{low} = \frac{N_M^J + M_{avg}}{S_M^J} + \frac{N_R^J \cdot (Sh_{avg}^{typ} + R_{avg})}{S_R^J} + Sh_{avg}^1 - Sh_{avg}^{typ} \quad (1)$$

where N_M^J is the number of map tasks, N_R^J is the number of reduce tasks, S_M^J is the number of map slots, S_R^J is the number of reduce slots and the tuple $(M_{avg}, R_{avg}, Sh_{avg}^1, Sh_{avg}^{typ})$ represents the performance invariants, for each MapReduce phase, extracted from the job profile. The equation for T_J^{up} can be written in a similar form and is detailed in Verma et al. [24]. It was reported that the average of lower and upper bounds (T_J^{avg}) is a good approximation of the job completion time, so we chose the upper bound as a conservative approach, avoiding the underestimation cases.

The algorithm implemented for the MapReduce Job Adaptor is presented in Algorithm 1. It starts by receiving the number of map and reduce tasks (Nm, Nr), and a profile p of the MR job to be executed in the cluster. It also gets information from the RMS, such as the list of free slots in the queue and maximum number of nodes and time that can be allocated in the queue. These limits in the number of nodes and time are usually imposed by HPC cluster administrators in order to enforce a fair sharing of resources between users.

Figure 2 presents an example of RMS queue of an HPC cluster with six jobs (A to F) scheduled in the queue. In this example, function $getFreeSlots()$ would return four free slots that could be used to execute the MR job. Slot 1 starts at time 10 with 25 nodes and maximum duration of 2. Slot 2 starts at time 13 with 50 nodes and maximum duration of 2. Slot 3 starts at time 16 with 25 nodes with no maximum duration, and slot 4 starts at time 17 with all cluster nodes and has also no maximum duration.

Variables $maxNodes$ and $maxTime$ receive the maximum number of nodes and amount of time that can be requested to the RMS. The turnaround variable, which stores the turnaround time of the best solution found by the algorithm, is initialized with a big number. After that, the algorithm starts testing each free slot in the RMS queue to verify if the MR job would fit on it while minimizing the turnaround time. Since the number of nodes can change for each slot, the execution time of the MR job needs to be estimated again. The execution time of the MR job is estimated using its parameters (number of map and reduce tasks, and job profile) and $numNodes$. The algorithm finishes with parameters *nodes* and *time* to be used in the job submission to the RMS which minimizes the turnaround time of the MR job. The turnaround time is calculated using the slot start time and the estimated execution time of the MR job subtracted by the current time (indicated by NOW in the algorithm).

4 Evaluation

In order to evaluate the proposed algorithm, we used a simulator based on the SimGrid toolkit [3], which provides abstractions and functionalities for the simulation of parallel and distributed systems, such as HPC clusters. We simulated a cluster of 128 nodes with 2 cores each (for the MapReduce experiments, we defined 1 map and 1 reduce task slot per node). Cluster's resources were managed by a RMS that allows users to submit jobs. We also simulated a stream of job submissions, where each job requires a number of nodes to be allocated for a particular amount of time.

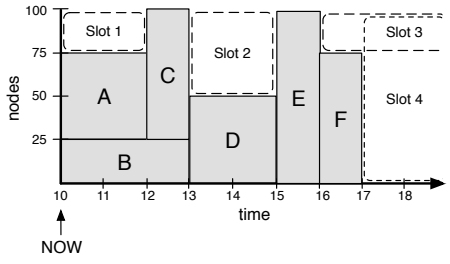

Fig. 2. Example of the queue of a Resource Management System

Algorithm 1 MapReduce Job Adaptor internal functioning

```
(Nm, Nr) ← Number of map and reduce tasks of MR job
p ← Job profile of MR job
freeSlotsList ← getFreeSlots()
maxNodes ← Maximum number of nodes allowed for allocation in the cluster
maxTime ← Maximum time allowed for allocation in the cluster
turnaround ← BigNumber
for all freeSlot in freeSlotsList do
  startTime ← getStartTime(freeSlot)
  slotDuration ← getSlotDuration(freeSlot)
  slotDuration ← MIN(slotDuration,maxTime)
  numNodes ← getNumberOfNodes(freeSlot)
  numNodes ← MIN(numNodes,maxNodes)
  execTime ← estimateJobExecutionTime(p, Nm, Nr, numNodes)
  newTurnaround ← startTime + execTime − NOW
  if execTime <= slotDuration and newTurnaround < turnaround then
    nodes ← numNodes
    time ← execTime
    turnaround ← newTurnaround
  end if
end for
return (nodes, time)
```

The simulated RMS implements the Conservative Backfilling (CBF) [8] algorithm. The CBF algorithm enables backfilling and is a representative of the algorithms running in deployed RMS schedulers today. The main idea of CBF is that an arriving job is always inserted in the first free slot available in the schedulers queue, which offers an upper-bound to the job start time. Every time a new free slot appears, the scheduler sweeps the entire queue looking for jobs that can be brought forward without delaying the start of any other job in the queue. This means that at any time it is possible to obtain the list of available free slots in the scheduler's queue. We use this feature to provide input for the algorithm described in the previous section.

We used a naive algorithm as a baseline for comparison purposes. It consists of allocating a number of nodes, based on the number of map and reduce tasks, during a fixed amount of time, defined as the maximum allowed amount of time per request. This is the case when there is no information about the MapReduce application and the scheduler's queue state. A similar approach is used by systems such as Hadoop On Demand [1] and myHadoop [13], yet in those systems the user has to specify the required number of nodes. The algorithms were compared in terms of average job turnaround time (interval between the submission of a job and its completion) and average system utilization. The number of simulations was defined in order to provide a confidence level of 95% with an error less than 5%.

Table 1. Distribution of job sizes in Facebook workload (based on Zaharia et al. [26]).

Bin	# Map Tasks	# Reduce Tasks	% Jobs at Facebook
1	1	0	39%
2	2	0	16%
3	10	3	14%
4	50	0	9%
5	100	0	6%
6	200	50	6%
7	400	0	4%
8	800	180	4%
9	2400	0	3%

To simulate a stream of job submissions for the users of an HPC cluster, we used two different approaches. The first was to simulate a synthetic workload based on a widely used model by Lublin et al. [14], which is one of the most comprehensive and validated batch workload models in the literature. Basically, it uses two gamma distributions to model the job inter-arrival time (depending on the time of day), a two-stage uniform distribution to model the job sizes and a two-stage hyper-gamma distribution to model the runtime of jobs.

We also used real-world workload traces from the Parallel Workloads Archive [2] as input to our simulation. This archive contains log information regarding the workloads on parallel machines, such as HPC clusters. We chose traces from the San Diego Supercomputer Center SP2 (SDSC SP2), which is a well-known and widely studied workload. SDSC SP2 workload has 128 nodes and 73,496 jobs, spanning 2 years from July 1998 to December 2000.

Unfortunately, there is not yet any such workload archive publicly available for MapReduce jobs. However, recent publications [25, 26, 4] have reported workload characteristics for MapReduce clusters in production at Google, Facebook and Yahoo!. We used the detailed description of a Facebook workload, provided by Zaharia et al. [26], to create a synthetic MapReduce workload. This workload comes from a Hadoop cluster, in production at Facebook in October 2009, with 600 nodes running about 7,500 jobs per day.

The Facebook workload used in our experiments is distributed in 9 bins as summarized in Table 1. As can be observed, most jobs in Facebook's workload are small. However, in the original workload, jobs in the last bin range from 1,501 to 25,000 maps. We chose 2,400 maps as our representative for this bin to make it fit in the HPC cluster simulated in our experiments. The job inter-arrival times is roughly exponential with a mean of 14 seconds. We defined map and reduce tasks duration as N(60,20) and N(120,30) respectively, where N(μ, σ) is the normal distribution with a mean μ and standard deviation σ.

The first experiment performed aims to evaluated the impact of the proposed algorithm in the job performance, in terms of average turnaround time and system utilization, for an HPC cluster running a mixed workload of HPC and MR jobs. We simulated one hour of HPC job submissions (around 400 jobs,

since the mean inter-arrival time in the so-called "peak hour" of the Lublin et al. model is roughly 5 seconds) mixed with one hour of MR job submissions (around 300 jobs).

Table 2 compares the results of the proposed algorithm (Adaptor) against the naive algorithm for each workload (HPC-only, MR-only and mixed HPC+MR). The proposed algorithm obtained shorter average turnaround time and improved utilization in all cases. For the MR-only workload, the use of the adaptor algorithm reduced the average turnaround time in 40%. For the mixed workload (HPC + MR), the overall average turnaround time was reduced in approximately 15%. However, the average turnaround time of the MR jobs in the mixed workload changed from 31776 (using naive algorithm) to 8616 seconds, which represent a reduction of 73%.

Table 2. Average job turnaround time and system load for each algorithm using Lublin et al. model (HPC) and Facebook (MR) workloads.

Workload (job type)	HPC	MR		HPC + MR	
Algorithm		Naive	Adaptor	Naive	Adaptor
Avg Utilization (%)	88.9	68.5	93.7	87.5	93.3
Avg Turnaround (s)	9126	6151	3709	13680	11512

To evaluate the influence of MR job sizes in our algorithm, we conducted experiments for each bin in Facebook's workload. Figure 3 shows the obtained results in terms of average job turnaround time. The adaptor algorithm outperformed the naive approach regardless the job bin. However, the adaptor algorithm performed better for bins with smaller job sizes. This happens because small job length cause more opportunity for backfilling. We believe that it is a positive characteristic, since the first 4 bins represent approximately 80% of the jobs in Facebook workload. Moreover, similar job size distribution can be seen in workloads from Google [25] and Yahoo! [4].

In order to evaluate the adaptor algorithm with different system loads, we conducted experiments varying the inter-arrival time of job submissions. The peak hour model by Lublin et al. produces mean inter-arrival time of 5.01 seconds, which is the mean of a Gamma distribution with $\alpha = 10.23$ and $\beta = 0.49$. Thus, different HPC load characteristics were simulated varying the value of α from 4 to 60, giving inter-arrival times between approximately 2 and 30 seconds. Similarly, different inter-arrival times for MR jobs were obtained by varying the mean in the exponential distribution described earlier. The results are shown in Figure 4. In both cases, the adaptor algorithm performed better regardless of the inter-arrival time.

Finally, to evaluate the performance of the adaptor algorithm using a real-world HPC workload, we chose a day-long trace from SDSC SP2 and used it along with 1,000 MR jobs as input for our simulation. Table 3 shows the results. The adaptor algorithm performed better in all cases. In this experiment, we also

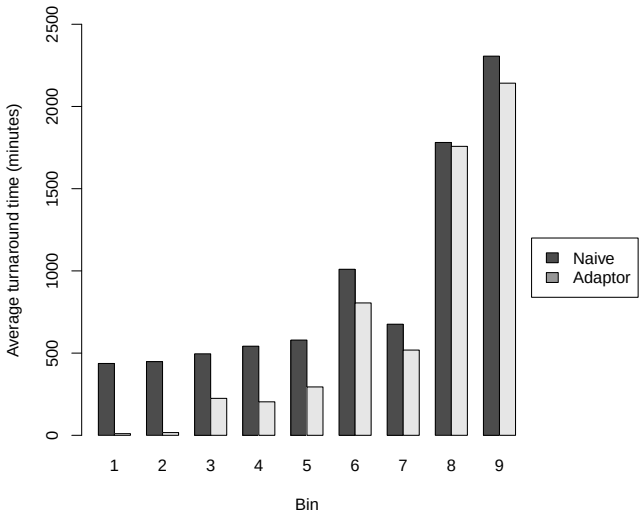

Fig. 3. Average job turnaround time for each bin in Facebook workload.

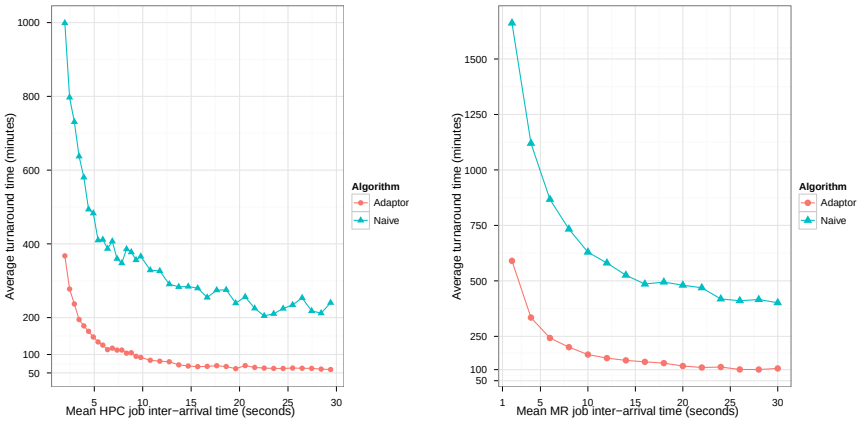

Fig. 4. Average turnaround time for a mixed workload varying (a) the mean job inter-arrival time of the HPC workload and (b) the mean job inter-arrival time of the MR workload.

observed that HPC and MR workloads are quite different. The HPC traces used to have few jobs with long running times, while MR have many jobs with short running times. This reinforces our argument that one should able to use an HPC cluster to run both HPC and MR jobs and that it can exploit unused resources.

Table 3. Average job turnaround time and system load for each algorithm using a trace from SDSC SP2 and Facebook workload.

Workload (job type)	HPC	MR		HPC + MR	
Algorithm		Naive	Adaptor	Naive	Adaptor
Avg Utilization (%)	52.5	83.4	89.4	56.3	68.4
Avg Turnaround (s)	16198	22602	10269	99629	19288

5 Conclusion and Future Work

MapReduce has gained attention by the HPC community, but it is still not trivial how HPC clusters can be exploited to execute such kind of job along with HPC applications. HPC clusters present some characteristics that conflict with the MapReduce model, such as the process used to submit jobs. This paper presented the MR Job Adaptor, a module that customizes regular MR jobs for submission in HPC clusters. MR Job Adaptor estimates the execution time of the job using an MR Job Profile and tests the available slots in the Resource Management System queue in order to allocate one that results in minimal turnaround time. The experiments performed to evaluate the module demonstrated that, besides minimizing the job turnaround time, it also exploits unused resources in the cluster.

As future work, we intend to evaluate other characteristics of the MR to enhance the algorithm used by the MR Job Adaptor. We believe that using a single cluster for HPC and MR jobs can be beneficial for both users and cluster administrators.

References

1. Apache Hadoop on Demand (HOD) (2012), http://hadoop.apache.org/common/docs/current/hod_scheduler.html, accessed on February 2012.
2. Parallel Workloads Archive (2012), http://www.cs.huji.ac.il/labs/parallel/workload/, accessed on February 2012.
3. Casanova, H.: Simgrid: A toolkit for the simulation of application scheduling. In: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001). Brisbane, Australia (May 2001)
4. Chen, Y., Ganapathi, A., Griffith, R., Katz, R.H.: The case for evaluating mapreduce performance using workload suites. In: MASCOTS. pp. 390–399. IEEE (2011)

5. De Rose, C.A.F., Ferreto, T., Calheiros, R.N., Cirne, W., Costa, L.B., Fireman, D.: Allocation strategies for utilization of space shared resources in bag of tasks grids. Future Generation Computer Systems 24(5), 331–341 (May 2008)
6. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
7. Ekanayake, J., et al.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 810–818. HPDC '10, ACM, New York, NY, USA (2010)
8. Feitelson, D.G., Mu'alem Weil, A.: Utilization and predictability in scheduling the IBM SP2 with backfilling. In: 12th Intl. Parallel Processing Symp. (IPPS). pp. 542–546 (Apr 1998)
9. Fox, G., et al.: Parallel data mining from multicore to cloudy grids. Proceedings of HPC 2008 (2011)
10. Gropp, W., Lusk, E., Skjellum, A.: Using MPI Portable Parallel Programming with the Message Passing Interface. The MIT Press (1994)
11. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R., Shenker, S., Stoica, I.: Mesos: Flexible resource sharing for the cloud. USENIX ;login: (August 2011)
12. Isard, M., et al.: Dryad: distributed data-parallel programs from sequential building blocks. Proceedings of EuroSys 2007 (Jan 2007)
13. Krishnan, S., Tatineni, M.: myhadoop-hadoop-on-demand on traditional hpc resources. sdsc.edu (2011), http://www.sdsc.edu/ãllans/MyHadoop.pdf
14. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. J. Parallel & Distributed Comput. 63(11), 1105–1122 (Nov 2003)
15. Middleton, A.: Data-intensive technologies for cloud computing. Handbook of Cloud Computing (Jan 2010)
16. Oracle: Oracle Grid Engine, previously known as Sun Grid Engine (SGE) (2012), http://www.oracle.com/technetwork/oem/grid-engine-166852.html, accessed on February 2012.
17. Schadt, E., Linderman, M., Sorenson, J.: Computational solutions to large-scale data management and analysis. Nature Reviews (Jan 2010)
18. Sehrish, S., et al.: Mrap: a novel mapreduce-based framework to support hpc analytics applications with access patterns. Proceedings of HPDC '10 pp. 107–118 (2010), http://doi.acm.org/10.1145/1851476.1851490
19. Srirama, S., Jakovits, P.: Adapting scientific computing problems to clouds using mapreduce. Future Generation Computer Systems (Jan 2011)
20. Team, A.H.: Apache hadoop web site (2011), http://hadoop.apache.org., accessed on February 2012.
21. Team, A.H.: Hamster: Hadoop and mpi on the same cluster (2011), https://issues.apache.org/jira/browse/MAPREDUCE-2911., accessed on February 2012.
22. Top 500: Top 500 Supercomputers Site (2012), http://www.top500.org, accessed on February 2012.
23. TORQUE: TORQUE Resource Manager (2012), http://www.clusterresources.com/products/torque-resource-manager.php, accessed on February 2012.
24. Verma, A., Cherkasova, L., Campbell, R.H.: Aria: automatic resource inference and allocation for mapreduce environments. Proceedings of ICAC '11 pp. 235–244 (2011)
25. Wang, G., et al.: Towards synthesizing realistic workload traces for studying the hadoop ecosystem. In: MASCOTS. pp. 400–408. IEEE (2011)
26. Zaharia, M., et al.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Morin, C., Muller, G. (eds.) EuroSys. pp. 265–278. ACM (2010)