2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

A Map-Reduce Based Framework for Heterogeneous Processing Element Cluster Environments

Yu Shyang Tan[1,3], Bu-Sung Lee[1,3], Bingsheng He[1], Roy H.Campbell[2]

[1]School of Computer Engineering
Nanyang Technological University
Singapore
{tanys,ebslee,bshe}@ntu.edu.sg

[2]Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
rhc@illinois.edu

[3]Service Platform Lab
HP Labs Singapore
Singapore
{yu-shyang.tan, francis.lee}@hp.com

***Abstract* - In this paper, we present our design of a Processing Element (PE) Aware MapReduce base framework, Pamar. Pamar is designed for supporting distributed computing on clusters where node PE configurations are asymmetric on different nodes. Pamar's main goal is to allow users to seamlessly utilize different kinds of processing elements (e.g., CPUs or GPUs) collaboratively for large scale data processing. To show proof of concept, we have incorporated our designs into the Hadoop framework and tested it on cluster environments having asymmetric node PE configurations. We demonstrate Pamar's ability to identify PEs available on each node and match-make user jobs with nodes, base on job PE requirements. Pamar allows users to easily parallelize applications across large datasets and at the same time utilizes different PEs for processing different classes of functions efficiently. The experiments show improvement in job queue completion time with Pamar over clusters with asymmetric nodes as compared to clusters with symmetric nodes.**

Keywords - MapReduce, GPGPU, Heterogeneous resource framework

I. INTRODUCTION

In the face of growing needs for computing power, various hardware accelerators are investigated as potential candidates for replacing or complementing the CPU. As shown in [1-5] and many other studies, an increasing number of researchers are using hardware accelerators such as Graphics Processors (GPUs) and CELL for processing. This is due to the huge potential performance gain that can be provided by these accelerators. Researchers have also begun looking at utilizing such accelerators in clusters to process their applications [6, 7]. The rising popularity of these accelerators is also evident from the growing presence of clusters equipped with accelerators in the top 500 Super Computers list [8]. With the increasingly widespread adoption of hardware accelerators, it will not be surprising to see more heterogeneous clusters in the near future. These heterogeneous clusters with different types of accelerators will enable users to use different processing elements (PEs) collaboratively, thereby harnessing the processing power of these accelerators for the computation of different tasks. Already there are studies looking into using CPU and GPGPU collaboratively [9, 10], in attempts to boost performance of matrix operations.

Clusters with nodes possessing different types of PE provide numerous benefits. On top of flexibility in choice of PEs to use, heterogeneous PE clusters can also provide a balance in cost-benefits when it comes to upgrading existing clusters with accelerators. In environments where users have different PE requirements, different partitions of a cluster can be upgraded to satisfy these diversified requirements. This avoids having a cluster-wide upgrade whenever there is a need for a new type of PE.

While clusters with heterogeneous node configurations can be beneficial, most existing distributed frameworks such as Hadoop *cannot* support these types of clusters. Those frameworks are not designed to support clusters with heterogeneous PE configurations. They lack the ability to gather the required information for the assignment of jobs with different PE requirements to the appropriate nodes in the cluster. As a result, users are either restricted to using only PEs that exists in all the nodes or to run their jobs in separate clusters which have the required PE in all the nodes. As such, frameworks which are able to detect different PE types on each node dynamically will become increasingly important in the future. Also, they should be able to make intelligent decisions when scheduling jobs to the nodes. This not only prevents jobs from failing due to mismatch in PE requirements of the job and available PEs on the node, but also improves the efficiency.

In this paper, we present a Processing Element Aware MapReduce base framework, Pamar. Pamar allows users to collaboratively use GPU and CPU PEs in their MapReduce jobs on clusters with asymmetry in GPGPU/CPU node configurations among nodes. Users do not have to worry about details like which nodes their jobs have to run on or how the data is to be partitioned. All those details are handled by Pamar. It automatically detects the type of PEs available on each node and scans for the PE requirements of submitted jobs with little or no user efforts. Pamar follows the MapReduce method; partitioning the data into chunks and distributing the tasks to the data for processing. The chunk granularity can be adjusted so that each data chunk can be fitted into the GPGPU accelerators. While the intention of Pamar is to facilitate the use of GPU accelerators in cluster settings, we do not constraint them to using pre-defined GPGPU functions like in [11]. Users are free to implement their algorithms to run on the GPUs. This enables greater flexibility in application development. In addition, we implemented HPE, a Heterogeneous Processing Element scheduler for Pamar. The algorithm handles the scheduling of jobs to satisfy the job requirements on PEs. As a proof of concept, we have integrated our design into Hadoop [12] and evaluated the prototype. Our experiments show that the prototype is able to yield up to 23% improvement in job queue completion time, compared with Hadoop.

This work is supported under the User and Domain Driven Data Analytics as a service Framework project, funded through the Thematic Strategic Research Program from A*STAR – SERC.

The remainder of this paper is structured as follows: In Section II, we introduce MapReduce, the underlying paradigm of Pamar. The design overview of Pamar and our prototype implementation is presented in Section III. In Section IV, we present the results of our experiments and discuss on related works in Section V. Finally, Section VI is the conclusion.

II. BACKGROUND

A. MapReduce and Hadoop

MapReduce [13] is a programming model from Google. The model comprises of two main functions, Map and Reduce functions. The Map function takes in a key/value pair and outputs an intermediate list of key/value pairs. The Reduce functions will than take all values associated to the same key and produce the final output list of key/values. Users are tasked with providing the Map and Reduce implementation.

Hadoop [12] is an open source implementation of MapReduce. It uses a JobTracker node to perform scheduling and management of user jobs. The JobTracker is also responsible for monitoring the TaskTracker nodes which are used to perform the raw computation. In additional to the JobTracker, Hadoop also uses another master node, the Namenode, for the management of data. With the Hadoop File system (HDFS) the Namenode manages the replication and distribution of the data blocks across all nodes within the cluster.

The main advantage of Hadoop over other existing distributed frameworks lies in allowing users to rapidly develop parallel and distributed applications with minimum knowledge on parallel programming required. The abstract Map and Reduce interfaces provided in Hadoop allow programmers to focus on application development. Hadoop abstracts non-application related tasks such as data partitioning, task scheduling, task failure detection and recovery and data redundancy from the users. This helps to ease the complexity of developing parallel and distributed applications for large dataset processing.

B. Hardware Architecture

Currently, most clusters are homogeneous in terms of PE type configurations between nodes within a cluster's environment. Meaning to say, if one of the nodes has only CPU processors as its PE, then all other nodes should only have CPU PE type. Likewise, if a node has a mix of 2 or more type of PEs, all other nodes should have the same mix of PE types. However, in practice, this is an inflexible solution. If the need for a new PE type arises, the whole cluster will have to be upgraded with the respective PE type, regardless of the user base. This might result in the PE being underutilized.

One solution to this problem is to perform a partial upgrade on the cluster, base on the demand for the required PE. However, this brings about the problem of current distributed frameworks not being able to support such cluster configurations. Current schedulers and frameworks are not designed to detect user job PE requirements and the available PE resources on each node. As such, schedulers will not be able to make correct scheduling decisions when scheduling task.

Pamar is designed to support cluster architectures such as the one shown in Figure 1. Pamar has built-in mechanisms for detecting the PE resource available on each individual node and for scanning the PE requirements for incoming user jobs. Both sets of information are then passed to the scheduler. With these information, the scheduler will be able to decide where the tasks can be scheduled to for smooth execution of the tasks.

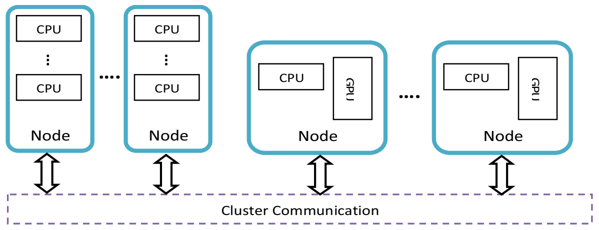


Figure 1.Heterogeneous cluster with asymmetry in node PE configurations

Pamar gives the flexibility of partially upgrading different portions of the cluster with different types of PEs to the administrators. This will encourage users to explore using more than one PE type in their jobs so as to increase efficiency. Pamar also adopts the MapReduce as the underlying programming paradigm, effectively allowing users of Pamar to easily write scalable parallel applications to process large sets of data. This is especially important in the current era where the size of readily available datasets is growing exponentially.

III. PAMAR

In this section, we first give a general overview of the various key components of Pamar. We describe the implementation of a prototype for Pamar which works for CPU/GPGPU heterogeneous clusters.

For simplicity in understanding, for the rest of this paper, we define 'job' and 'tasks' as follows: A job is a user submitted workload consisting of multiple tasks, each responsible for a portion of the workload that a job is required to complete. For a job to be considered completed, all tasks from the same job have to be completed successfully.

A. Detecting Available PE Resources

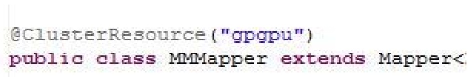
Before Pamar performs scheduling tasks with different PEs, it needs to detect the available PE types in each node within the cluster. This is achieved by scanning the nodes for the presence of PE types.

When a node is initialized, a scan is conducted to look for specific PE resources on the node. API libraries for the specific PEs are used in the scanning algorithm. The scan is executed on the node side through a start up script. This is to avoid placing additional load on the master server. By having the nodes execute the script themselves, it not only allows the simultaneous scanning of multiple nodes, but also allows new nodes to join the cluster at a later time independently. This can help avoid disruption to the master server even when nodes join the cluster during operation time. Once the scan is complete, a list of available PEs is made for each node and is sent to the master server. The information is stored in the master server and used during the scheduling phase.

B. Scanning of Job Requirements

In scheduling, Pamar also tries to scan for the PE requirements of each task. Remembering our definition of task stated previously, the task can be a Map or Reduce function implemented by the user. Our objective here is to obtain the information needed with as little user intervention as possible. To achieve this, a dual level code scanning scheme is used. Figure 3 shows the flow of the whole scanning of job requirement process. This is done on the client side to maintain scalability and prevent bottlenecks at the master server.

The first level looks at allowing users to inform Pamar what is required for each task through built-in APIs. In our prototype, a custom annotation class, shown in Figure 2, is implemented to allow users to specify to Pamar the PE requirements.



```
@ClusterResource("gpgpu")
public class MMMapper extends Mapper<
```

Figure 2.Custom Annotation class, *ClusterResource*, used to denote PE requirements at task level

The client component of Pamar than scans through the source code of the task functions in attempts to pick up such annotation labels. If successful, the task requirement is marked based on the label. Otherwise, a second level of scanning is executed by the client. The client will search through the task's source codes or compiled class files for relevant keywords to identify invocations of PE specific API libraries. Such traces can help Pamar to decide if the relevant PEs are being used in the tasks. As this method of scanning for keywords is not fool-proof, it is only utilized in the absence of an annotation label. We tested the source code scanning feature using several different programs written in Java and JCUDA and achieve 100% accuracy in mapping tasks to the relevant PEs.

By having two levels of scanning, Pamar can still get some clue on the PE requirements of tasks even if the annotation labels are not used.

C. Scheduling of Tasks

With the two pieces of information at hand, Pamar makes decisions on which task gets assigned to the nodes, without having task failures due to mismatch between the task PE requirements and the PE available on the node. However, during our initial testing of the Pamar framework, we found that using the Hadoop default scheduling algorithm produces unstable results. We will discuss this in more detail in the experimental results section.

After our analysis on the initial results, we found that the GPGPU tasks generally finish faster than the CPU tasks. This gives the idea of finishing the "shorter" tasks first so that the idle resources can be used to process the "longer" tasks. We also found that there is usually a waiting time incurred by tasks that requires the GPGPUs if it is queued behind a CPU task. This is so as the GPGPU tasks must run on a GPGPU enabled node. However, this is not so for the CPU tasks as each node comes equipped with a CPU processor by default. Therefore, if the CPU task is queued in front, than it will most likely occupy the GPGPU enabled node too. In such situations, even if some of the tasks complete first, freeing up nodes to take in new tasks, if the nodes are not GPGPU enabled, then the GPGPU task will have to continue waiting. This does not apply to the CPU tasks as they can run on any nodes within the cluster by default.

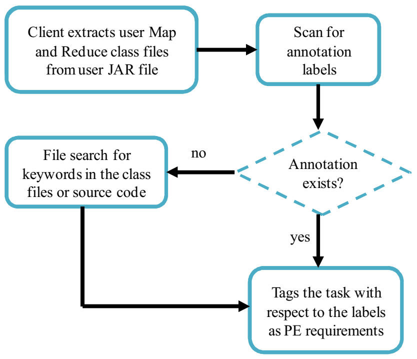

Client extracts user Map and Reduce class files from user JAR file
Scan for annotation labels
Annotation exists?
no
File search for keywords in the class files or source code
yes
Tags the task with respect to the labels as PE requirements

Figure 3. Flowchart of the scanning of job requirement process

Base on the observations, we made 2 basic assumptions;

1. GPGPU tasks generally execute faster than their CPU equivalents due to the ability to utilize more parallel threads.
2. GPGPU tasks can only execute on nodes with GPUs while CPU tasks can execute on all nodes as each node comes with a CPU PE at its core.

Taking into considerations the assumptions, we developed a scheduler, the Heterogeneous PE scheduler (HPE), to complement Pamar in the task scheduling process. At its core, the HPE scheduling algorithm uses the First-Come-First-Serve (FCFS) policy when scheduling tasks to nodes. However, priority is given to tasks that require the GPGPU PEs in the event the node requesting for task is a GPGPU enabled node. Algorithm 1 shows the pseudo code for the HPE scheduler.

Algorithm 1: HPE scheduler for Pamar

```
1   initialize $skipcount \rightarrow 0$, $jobGpu \rightarrow$ false, $nodeGpu \rightarrow$ false
2   get node profile from master server
3   $nodeGpu \rightarrow$ get GPGPU flag of node profile
4   check no. of available compute slots for target node
5   compute total no. of tasks to request for
6   foreach $j \in jobQueue$ & $skipcount < 3$
7     $jobGpu \rightarrow$ get GPGPU flag of job profile
8     if $jobGpu ==$ true & $nodeGpu ==$ true
9       AssignTask(node profile, $j$)
10    else if $jobGpu ==$ true & nodeGpu == false
11      continue foreach loop
12    else if $jobGpu ==$ false & $nodeGpu ==$ true
13      $skipcount \rightarrow skipcount + 1$
14      continue foreach loop
15    else AssignTask(node profile, $j$)
16  end foreach
17  //Cannot find any GPGPU tasks
18  if skipcount == 3
19    $j \rightarrow$ first job in queue
20    AssignTask(node profile, $j$)
21  end if
```

The **AssignTask** function will try to select a task which is processing data that is local to the node. If such task cannot be found, it will then randomly assign a task from within the job.

The algorithm attempts to match a task which requires the GPGPU to a GPGPU enabled node. Keeping in mind that a GPGPU task has to run on a GPGPU enabled node, if a GPGPU task encounters a non-GPGPU enabled node, the scheduler will try to look for a CPU task from other jobs within a queue. On the other hand, if a GPGPU enabled node encounters a CPU task, the algorithm will attempt to search for a GPGPU task from the next three jobs in the front of the queue. If no such task can be found, it will schedule the CPU task to the node. The limit of skipping three jobs is to prevent starvation of CPU jobs in the event where there is a constant incoming stream of jobs that requires the GPGPU.

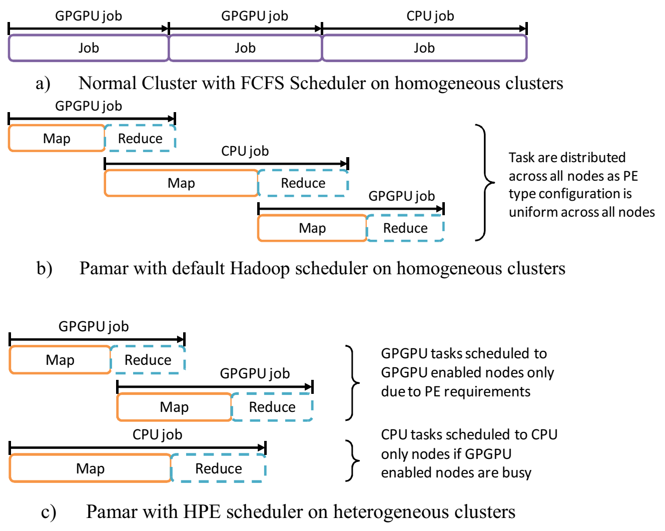
Figure 4 shows the different scenarios when running the FCFS policy scheduler in normal circumstances, using the default Hadoop scheduler on homogeneous clusters and when using HPE on heterogeneous clusters. It can be seen that jobs with different PE requirements do not have to wait for each other as the HPE scheduler takes into consideration assumption 2 when performing task scheduling. It results in jobs with different PE requirements being executed in parallel on nodes having different PE configurations as illustrated in Figure 4c. This differs from other schedulers, such as the Hadoop default scheduler, where jobs have to wait for the previous job to complete before proceeding, as shown in Figure 4b. Note that in MapReduce frameworks, Map tasks can run in parallel with Reduce tasks as they run on separate task slots, thus the overlapping in task execution shown in Figure 4b.

Up till now, our discussions have been on general heterogeneous environments, not limited to only GPGPUs and CPUs, however, the assumptions made here and the implementation of the HPE are both targeted at the prototype implementation of Pamar. As such, only CPU and GPGPU PEs are mentioned in this part of the discussion. In the future Pamar can be extended to support other types of PEs by adding in the relevant device flags into the framework. Figure 5 illustrates the design overview of the architecture of Pamar.

D. *Adopting the MapReduce paradigm*

Pamar is designed to be based off the MapReduce paradigm for several factors. The first factor is the handling of data and the ease of processing large datasets in parallel. The way data is partitioned into blocks and having tasks parallelized across those blocks for parallel processing allows for easy managing and processing of large sets of data. In terms of implementation, such mechanisms can be easily implemented into the framework, thus providing automatic data management features to the users.

The second factor is that hardware accelerators usually have limited amount of onboard memory space and that different versions of the devices have differing amount of such memory spaces. This leads to a constraint on how much data can the accelerators process at one time. By partitioning the data into smaller blocks, it helps to make large data processing more manageable by the memory limited hardware accelerators. Users will not have to worry about whether the data block to be processed by the GPGPU task can be fitted onto the memory of the device. The framework can also easily accommodate accelerators with non-uniform amount of memory space via tweaking the size of the data blocks.



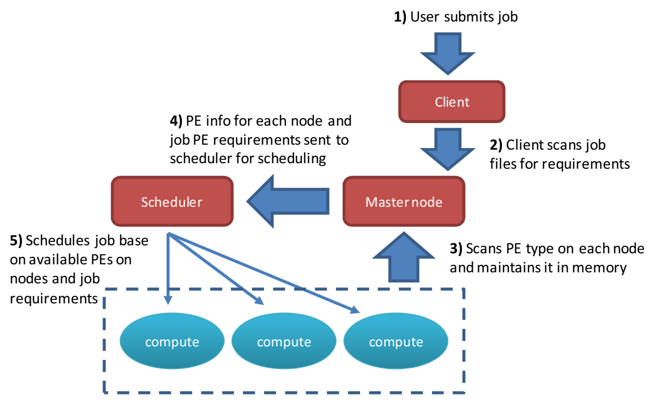
GPGPU job
GPGPU job
CPU job
Job
Job
Job
GPGPU job
Map
Reduce
CPU job
Map
Reduce
GPGPU job
Map
Reduce
Task are distributed across all nodes as PE type configuration is uniform across all nodes
GPGPU job
Map
Reduce
GPGPU job
Map
Reduce
GPGPU tasks scheduled to GPGPU enabled nodes only due to PE requirements
CPU job
Map
Reduce
CPU tasks scheduled to CPU only nodes if GPGPU enabled nodes are busy

a) Normal Cluster with FCFS Scheduler on homogeneous clusters

b) Pamar with default Hadoop scheduler on homogeneous clusters

c) Pamar with HPE scheduler on heterogeneous clusters

Figure 4. Illustration of job scheduling in different scenarios



1) User submits job
Client
2) Client scans job files for requirements
Master node
3) Scans PE type on each node and maintains it in memory
4) PE info for each node and job PE requirements sent to scheduler for scheduling
Scheduler
5) Schedules job base on available PEs on nodes and job requirements
compute
compute
compute

Figure 5. Design overview of the architecture of Pamar

The last factor is in allowing the use of different types of PE in a collaborative manner in a single job. By dividing a job into multiple phases such as the Map and Reduce phase, it allows the user to better plan and implement which portion of the job uses what type of PE. Such segmentation also makes scheduling of the tasks much easier. In cases where only a small portion of the job uses the GPGPU for a short time while the rest of the job runs for a long time, such jobs will not end up hogging the GPGPU PEs for a long time unnecessarily.

E. *Implementing the Prototype*

In this section, we describe the modifications made to Hadoop, for the implementation of a proof of concept prototype for Pamar. While what we describe in previous sections is a general architecture design, it can be implemented into most MapReduce frameworks such as Disco[14]. Hadoop is chosen due to its widespread use and easy availability of source code. Figure 6 shows the integration of Pamar into Hadoop. The components with solid shaded backgrounds and solid borders represent the components which have been modified to suit Pamar's needs. On the other hand, the components with crisscross background shades and dotted borders are components new to the framework.

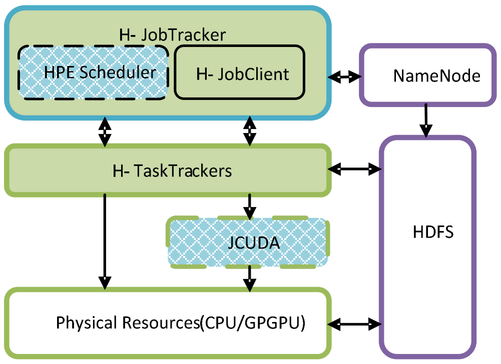


Figure 6. Components of Prototype of Pamar

Of the two new components, the HPE scheduler is the task scheduler module mentioned in the previous section, implemented by us and is responsible for the scheduling of tasks to the nodes. The JCUDA[15] module on the other hand is the GPGPU API library used both in the scanning of GPGPU PEs on nodes and used by users for interfacing with the GPGPUs. In future, if new PE types are to be supported, the APIs of the respective PEs can be added here for Pamar to support those PEs.

The H-JobTracker module is essentially the master server of Hadoop. This module is modified to incorporate the storing and allowing for the retrieval of the node configurations by the scheduler. In additional, the master server also coordinates the passing of the flag status of PE requirements of the user jobs from the client to the scheduler. The H-TaskTracker is the node side script that is responsible for initializing the node it is deployed on. The PE scanning mechanism is incorporated into the H-TaskTracker script. The mechanism will scan for the presence of GPGPU devices on a node and relays this information to the H-JobTracker. These two modules work hand in hand to detect and store each node's PE configuration within a cluster.

Lastly, the H-JobClient is the client module that accepts user jobs. The scanning of user job files for task PE requirements is done here. Once done scanning, the appropriate flags in the job configuration file is set by the module and passed on to the master server for task scheduling.

IV. EXPERIMENTAL EVALUATION

A. Experimental Setup

To evaluate Pamar, we setup and deploy Pamar on three different cluster configurations using resources from Amazon EC2 cloud. Two main types of instance are being used, the Cluster Compute instance to represent the *CPU only node* and the Cluster GPU instance to represent the *GPGPU enabled node*. The detailed specifications of these two instance types can be found at [16]. Table 1 shows the three cluster configurations setup in the EC2 environment.

We set the amount of Map task slots on the *GPGPU enabled node* to 3 as the GPU can only run 1 task at a time. The setting of 3 tasks allows the node to overlap the loading of data of the 2nd and 3rd task with the execution of the 1st task. Once the first task completes, the 2nd task can execute and the 3rd task would give the node time to request and prepare for new tasks execution, thus fully utilizing the node.

Table 1. Summary of EC2 cluster configurations

Config type	No. of CPU nodes	No. of GPGPU nodes	No. of GPGPU enabled map task slots	No. of CPU map task slots	No. of GPGPU enabled reduce task slots	No. of CPU reduce task slots
E1	8	0	0	32	0	8
E2	0	8	32	0	8	0
E3	2	6	24	8	6	2

Three applications were used to substitute for jobs with different PE requirements in our experiments;

Matrix Multiplication: The application looks at multiplying two 6000 by 6000 square matrices, M and N together. Each Map task is given an equal sub-portion of matrix M and is responsible for computing that sub-portion. The Reduce function merges back the results from each Map task to form the final resulting matrix. We implement the Maps to run on the GPGPU, utilizing its massive parallel threads feature.

Word Count: This benchmarking application from the Apache Hadoop examples is used here to count the number of occurrences of each word that appears in a set of text files.

DNS log analysis: This application extracts data from log files of the root DNS servers, then uses the data to compute and determines the performance of the root DNS servers based on a custom defined performance matrix [17]. The Map tasks processes each file individually and performs the computations required on the GPGPU. The Reduce task then merges the sub-results to form the final output.

The Matrix Multiplication and DSN log analysis applications are not optimized to run on the GPGPUs. This is so as our main focus is on a framework that is able to intelligently schedule jobs with different PE requirements to the corresponding nodes automatically in clusters with heterogeneous PEs and node configurations. The data to be processed and the application used in each experiment are kept constant. The reason is on the response time for the jobs to run on Pamar in the different cluster configurations.

B. Homogeneous vs. Heterogeneous

In this experiment, we look at how Pamar performs in the different cluster configurations. We construct a queue with jobs having different PE requirements, using the 3 mentioned applications. The jobs are submitted in the following order; Matrix Multiplication, DNS log and the Word Count application. We also used a CPU only job queue as a baseline for comparison. In this queue, the Matrix Multiplication and DNS log applications are re-implemented to use only on the CPU for processing. This job queue is term as "CPU version" in the subsequent figures. Figure 7 shows the time it takes to complete each job and the entire job queue for each cluster configurations. The job timings are recorded from the time of submission till the time the job completes. For the timing recorded for the job queues, timing is taken from the submission of the first job to the queue till the completion of the last job in the same job queue. While the same Word Count application is used on all 3 cluster settings, we note that the difference in its completion timing across the different clusters. This is due to the different amount of GPGPU enabled nodes being utilized in the initial processing of the GPGPU tasks, thus reducing the number of nodes use in processing the Word Count application.

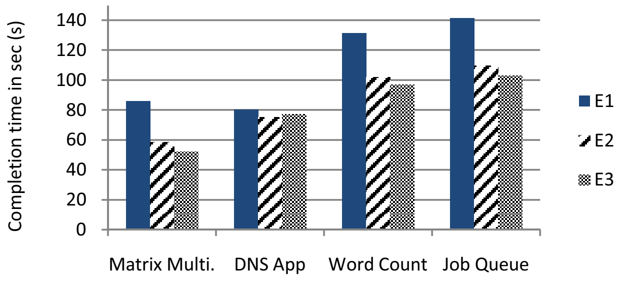


Figure 7.Completion time of each job and job queue completion for each cluster configuration

The main interest here is in the job queue completion timing. We argue that by having different nodes serving the various jobs with different PE requirements, it is possible to reduce the waiting time between jobs. This can be observed from the last set of data, the Job Queue results, in Figure 7. The heterogeneous cluster, E3, manage to complete the entire job queue first, among the three clusters. This is attributed to the HPE scheduler. Since HPE gives priority to GPGPU tasks, it will schedule the GPGPU tasks, i.e, the Map tasks of Matrix Multiplication and DNS log jobs, to the *GPGPU enabled nodes*. This leaves the *CPU only nodes* available to process the Word Count job. In this manner, jobs with different PE requirements are executed in parallel, as illustrated in Figure 4c. Compared to the homogeneous clusters E1 and E2, tasks from each job are distributed equally across all nodes for processing. Even though more nodes can be used in the processing of jobs, jobs at the back of the queue have to wait for the previous ones to finish, incurring longer waiting time. This is illustrated in Figure 4b.

In Figure 7, the gain in overall job queue completion time from the E3 compared to E2 is small. However this is so as there are only 3 jobs in the queue. Figure 8 shows the difference between job queue completion time of the homogeneous and heterogeneous clusters, increasing as the no. of jobs in the queue increases. At a job queue length of 9, Pamar can provide up to approximately 23% of improvement in terms of job queue completion time.

Having said so, one must also be careful when determining the ratio of the different nodes within a heterogeneous cluster. Compared to homogeneous cluster configurations, heterogeneous clusters with asymmetry node configurations will have lesser amount of specialized PEs to process jobs with requirement for the respective PEs. This might potentially lead to a longer processing time as there are lesser nodes to share the workload. This can be seen vaguely in the performance of the DNS application in Figure 7. The homogeneous cluster outperforms the heterogeneous cluster marginally as there are lesser nodes with GPGPU resources.

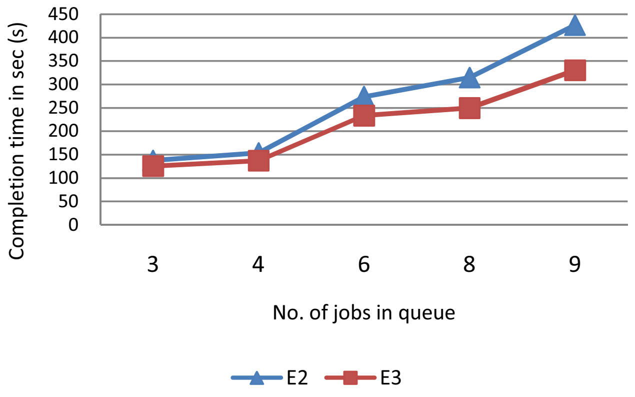


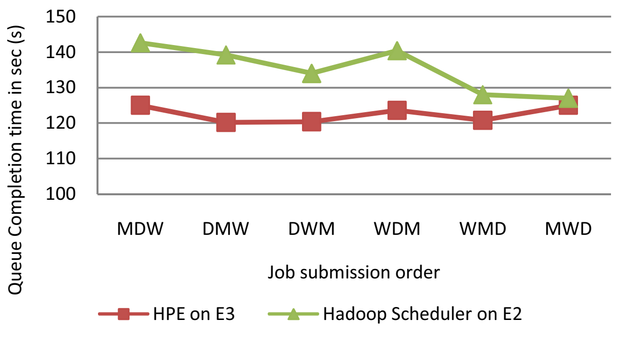
Figure 8.Job queue completion time for different no. of job submission between E2 (homo.) and E3 (hetero.)

C. Impact of jobs order

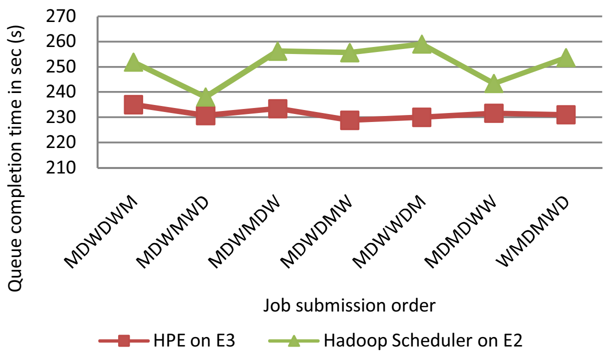
Next, we look at how job submission order will affect the job queue completion time in Pamar. In this experiment, we run Pamar on E2 to represent homogeneous cluster and E3 to represent heterogeneous cluster. We use the Hadoop's default scheduler on E2 as a comparison for HPE scheduler. We submit the Matrix Multiplication (M), DNS log analysis (D) and Word Count (W) applications in different order to the different scenarios and record the job queue completion time. The results are shown in Figure 9. Overall, heterogeneous clusters running Pamar with HPE shows better job queue completion time compared to homogeneous clusters running Pamar with Hadoop's default scheduler. This is attributed to how tasks are scheduled, as explained in the previous sections.

Figure 9 shows that the job queue completion time for the HPE scheduler is not affected by the order as compared to the Hadoop's default scheduler. The y-axis of the Figures in Figure 9 do not start from zero for clarity sake of the Figures. Recall the assumptions and scheduling decision for the HPE scheduler. We assume that GPGPU tasks generally complete execution faster than the CPU tasks due to its ability to utilize large amount of parallel threads. Thus giving priority to GPGPU tasks when nodes with GPGPU PEs requests for task can help to process jobs in the queue faster, minimizing the impact on other jobs in the queue. Due to the consideration of priority, the scheduling order becomes more or less the same regardless of the order of submission for the HPE scheduler. On the other hand, the Hadoop scheduler is very much affected by the order. Because in each node, the task slots for Map and Reduce tasks are separated, thus it is possible for Map and Reduce tasks from different jobs to run in parallel on a single node. As such, situations where the previous job has a long Reduce phase overlaps with the next job with a long Map phase but short Reduce phase will have better performance than one with the opposite properties. This applies even as the number of job submission increases, as shown in Figure 9b, where the number of jobs submitted is increased to 6.

In situations where influx of user jobs with different PE requirements is constant, HPE scheduler will provide a more stable execution time as compared to the Hadoop's default scheduler. It is true that one may obtain optimized results from proper ordering of jobs when using Hadoop default scheduler, as shown in Figure 9a. However, the order of job submissions on operational clusters is an uncertainty.



a) Queue of 3 jobs



b) Queue of 6 jobs

Figure 9.Job queue completion time for different job submission order in different scenarios

V. RELATED WORK

[18-20] are MapReduce frameworks that support GPGPU processing on standalone machines. [18] reduces complexity of GPGPU programming through generating GPGPU codes for the user. Mars[19] on the other hand uses APIs and pre-defined Map and Reduce function. The scheduler takes care of the transfer of data between the main memory and the device memory and also the invocation of functions on the GPGPU. MapCG[20] aims to enable users to write once, deploy on different architectures through a code abstraction layer. The compiler will compile the user codes into the relevant binary version for the respective PEs. Merge [11] is a library-oriented platform for heterogeneous multi-core system. The Merge compiler and runtime dynamically selects the best function variant from the library, for a given input and machine configuration, compiles it and sends it to the appropriate PE for processing. PLASMA [21] looks at compiling applications during runtime for different PEs and schedules the applications to run on the different PEs in parallel. Although [11, 18-20] demonstrates significant speed up when compared to CPU processors, the frameworks run on standalone machines with GPGPU PEs. [22] is an extension of Mars to work in clusters. However it only runs on homogeneous clusters with GPGPU PEs. On the other hand, Merge can be extended to support other types of accelerators by adding the respective function variants into the library. Such extensibility has been incorporated into Pamar. Compare to these works, Pamar provides heterogeneity support in a cluster setting.

[23-26] are some works that attempts to utilize multiple PEs in a distributed cluster setting. QP[23] is a multi-accelerator cluster with homogeneous node PE configuration. Each node is equipped with CPU, GPU and FPGAs PEs. Axel's[24] cluster configuration is similar to QP, but using FGPAs and GPUs PEs and is based off MapReduce. MGP [25] looks at providing programmers the flexibility of running their applications on the GPUs transparently by providing them with the MGP APIs. These frameworks uses multiple software such as Phoenix [27], MPI and Torque for multi-level task scheduling, adding complexity to the frameworks. They allow users to utilize different PEs collaboratively. Differing from those frameworks, [26] uses a single layer of scheduling but focus on using the GPGPU for computation. The CPU is used only for optimizing the transfer of the output results between the main memory and the GPGPU's device memory. The crucial difference with Pamar is that the mentioned frameworks are only capable of supporting clusters which have homogeneous node PE configurations. In comparison, Pamar targets at clusters with nodes having different PE configurations, without having code execution issues and requires little effort from users when deploying their applications.

Considerable amount of work have already been done in the field of heterogeneous resource scheduling. [28] investigates using CPU and GPGPU collaboratively for improving performance of compute intensive applications. Using the Anthill runtime environment, they evaluated two scheduling algorithms, FCFS and DWRR. The DWRR policy divides the events to be process and assigns a dynamic weightOn the other hand, Lei Wang et al[29] looks at first profiling the tasks on the CPU and GPU separately, then base on a model, decide whether to schedule the task to run on the CPU or GPU. However, doing so requires the application to be implemented in both CPU and GPGPU before any profiling can be done.

VI. CONCLUSION

In this paper, we presented Pamar, a MapReduce base framework for clusters with nodes having heterogeneous PE configurations. Pamar detects both the PE configurations on each node in the cluster and the PE requirements of user jobs and does the match-making of tasks via scheduling, for the user with little user effort required. We also showed how the HPE scheduler is implemented to complement the Pamar framework, increasing the stability and robustness of Pamar.

In the future, we envision that clusters will be equipped with different accelerators to cater to efficient processing of jobs with different properties. We plan to extend Pamar to support other accelerators such as CELL in heterogeneous clusters. Our focus in this work is on developing a framework that is capable of supporting node PE configuration heterogeneity across a cluster. Scheduler is not the main focus, as such for ease of implementing; the FCFS policy is chosen for the HPE. However, results shown in [28] clearly tells us that using FCFS is not sufficient in trying to achieve a highly efficient framework. Beyond performance optimizations, monetary efficiency of heterogeneous executions should be investigated[30, 31]. Therefore, improving the HPE scheduling algorithm will also be one of our future focuses.

REFERENCE

[1] Garnett Wilson and Wolfgang Banzhaf; "Deployment of CPU and GPU-based genetic programming on heterogeneous devices," in *GECCO'09*, Québec, 09.

[2] Chen Chen, Bertil Schmidt, Liu Weiguo, and Wolfgang Muller-Wittig; "GPU-MEME: Using Graphics Hardware to Accelerate Motif Finding in DNA Sequences," in *Third IAPR International Conference on Pattern Recognition in Bioinformatics*, Melbourne, Australia 2008.

[3] Bowen Zhang and Cornelis W. Oosterlee; "Option pricing with COS method on graphics processing units," in *IPDPS'09*.

[4] Svetlin Manavski and Giorgio Valle; "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics* vol. 9, 2008.

[5] Kanupriya Gulati and Sunil P. Khatri; "Accelerating Statistical Static Timing Analysis Using Graphics Processing Units," in *ASP-DAC09*, Yokohama, Japan, Jan 2009.

[6] Reza Farivar, Abhishek Verma, Ellick Chan, and Roy H. Campbell; "MITHRA: Multiple data Independent Tasks on a Heterogeneous Resource Architecture," in *Cluster09*, New Orleans, Louisiana, 2009.

[7] George Teodoro, Rafael Sachetto, Olcay Sertel, Metin N.Gurcan, Wagner Meira Jr., Umit Catalyurek, and Renato Ferreira; "Coordinating the use of GPU and CPU for Improving Performance of Compute Intensive Applications," in *Cluster 09*, Louisiana, 2009.

[8] TOP500.org. Available: http://www.top500.org/list/2010/06/100

[9] *MAGMA*. Available: http://icl.cs.utk.edu/magma/index.html

[10] Stanimire Tomov, Jack Dongarra, and Marc Baboulin; "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Comput.,* vol. 36, pp. 232-240, 2010.

[11] Michael D. Linderman, Jamison D. Collins, Hong Wang, and Teresa H. Meng; "Merge: A Programming Model for Heterogeneous Muti-core Systems," in *ASPLOS'08*, Seattle, Washington, USA, Mar 2008.

[12] *Hadoop*. (April). Available: http://hadoop.apache.org/common/docs/current/mapred_tutorial.html

[13] Jeffrey Dean and Sanjay Ghemawat; "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, Dec 2004.

[14] Spiros Papadimitriou and Jimeng Sun; "Disco: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining," in *International conference on Data Mining, ICDM'08*, Pisa, Italy, 2008.

[15] *JCuda - Java binding for CUDA*. Available: http://www.jcuda.org/jcuda/JCuda.html

[16] *Amazon EC2 instance type*. Available: http://aws.amazon.com/ec2/instance-types/

[17] Bu-Sung Lee, Yu Shyang Tan, Yuji Sekiya, Atsushi Narishige, and Susumu Date; "Availability and Effectiveness of Root DNS servers: A long term study," in *NOMS'10*, Osaka.

[18] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer; "A Map Reduce Framework for Programming Graphics Processors," in *STMCS08*, Boston.

[19] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang; "Mars: A MapReduce Framework on Graphics Processors," in *PACT'08*, Toronto, Ontario, Canada, Oct 2008.

[20] Chuntao Hong, Dehao Chen, Wenguang Chen, Weimin Zheng, and Haibo Lin; "MapCG: writing parallel program portable between CPU and GPU," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, Vienna, Austria, 2010, pp. 217-226.

[21] Sreepathi Pai, R.Govindarajan, and M.J. Tjhazhuthaveetil; "PLASMA: Portable Programming for SIMD Heterogeneous Accelerators," in *HPCA/PPoPP'10*, Bangalore, India, Jan 2010.

[22] Fang Wenbin; "Mars: Accelerating MapReduce with Graphics Processors," *IEEE Transactions on Parallel and Distributed Systems,* vol. 99, 2010.

[23] Michael Showerman, Jeremy Enos, Avneesh Pant, Volodymyr Kindratenko, Craig Steffen, Robert Pennington, and Wen-mei Hwu; "QP: A Heterogeneous Multi-Accelerator Cluster," in *10th LCI International conference on High-Performance Clustered Computing*, Boulder, Colorado, Mar 2009.

[24] Kuen hung Tsoi and Wayne Luk; "Axel: A Heterogeneous Cluster with FPGAs and GPUs," in *FPGA'10*, Monterey.

[25] Amnon Barak, Tal Ben-nun, Ely Levy, and Amnon Shiloh; *A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices*. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.173.678

[26] Jeff A. Stuart and John D. Owens; "Multi-GPU MapReduce on GPU Clusters," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, pp. 1068-1079.

[27] Richard M. Yoo, Anthony Romano, and Christos Kozyrakis; "Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System," in *IISWC09*, Austin, Tx, Oct 2009.

[28] George Teodoro, Rafel Sachetto, Olcay Sertel, Metin N. Gurcan, Wagner Meira Jr., Umit Catalyurek, and Renato Ferreira; "Coordinating the use of GPU and CPU for improving performance of compute intensive applications," in *Cluster Computing and Workshops CLUSTER'09*, New Orleans, Louisiana, 2009.

[29] Lei Wang, Yong-zhong Huang, and Xin Chen; "Task Scheduling of Parallel Processing in CPU-GPU Collaborative Environment," in *Computer Science and Information Technology ICCSIT'08*, 2008.

[30] Shadi Ibrahim, Bingsheng He, and Hai Jin; "Towards Pay-As-You-Consume Cloud Computing," in *Proceedings of the 2011 IEEE International Conference on Services Computing*, 2011, pp. 370-377.

[31] Hongyi Wang, Qingfeng Jing, Rishan Chen, Bingsheng He, Zhengping Qian, and Lidong Zhou; "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, Boston, MA, 2010, pp. 6-6.