In Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Melbourne, Victoria, Australia. May 2010.

Designing Accelerator-Based Distributed Systems for High Performance

M. Mustafa Rafique, Ali R. Butt
Department of Computer Science
Virginia Tech
Blacksburg, Virginia, USA
Email: {mustafa, butta}@cs.vt.edu

Dimitrios S. Nikolopoulos*
Institute of Computer Science
Hellas (FORTH)
GR-70013, Heraklion, GREECE
Email: dsn@ics.forth.gr

***Abstract*—Multi-core processors with accelerators are becoming commodity components for high-performance computing at scale. While accelerator-based processors have been studied in some detail, the design and management of clusters based on these processors have not received the same focus. In this paper, we present an exploration of four design and resource management alternatives, which can be used on large-scale asymmetric clusters with accelerators. Moreover, we adapt the popular MapReduce programming model to our proposed configurations. We enhance MapReduce with new dynamic data streaming and workload scheduling capabilities, which enable application writers to use asymmetric accelerator-based clusters without being concerned with the capabilities of individual components. We present an evaluation of the presented designs in a physical setting and show that our designs can provide significant performance advantages. Compared to a standard static MapReduce design, we achieve 62.5%, 73.1%, and 82.2% performance improvement using accelerators with limited general-purpose resources, well-provisioned shared general-purpose resources, and well-provisioned dedicated general-purpose resources, respectively.**

***Keywords*-Accelerator-based systems; heterogeneous clusters; resource management; programming asymmetric clusters; capability-aware task distribution**

I. Introduction

Multi-core processors can integrate several general-purpose cores (e.g. x86, PowerPC) and computational accelerators (e.g. SIMD processors and GPUs), yielding highly power-efficient and cost-efficient designs, with performance exceeding 100 Gflops [1]–[7]. These asymmetric accelerator-based processors are rapidly becoming commodity components for high-performance computing [8]–[17]. The commoditization of accelerator-based multi-core processors enables their deployment in large-scale clusters, as substitutes of less cost-effective alternatives.

The state of knowledge on the use of accelerator-based multi-core processors on large-scale clusters is limited. There is an inherent imbalance between general-purpose cores and accelerators in asymmetric settings. General-purpose cores are efficient in executing control-intensive code, therefore they tend to be employed primarily as controllers of parallel execution on and communication with accelerators. Accelerators on the other hand, are efficient in executing data-parallel computational tasks. Current approaches for designing and programming accelerator-based clusters are either ad hoc or specific to an installation [14], thus posing several challenges when applying to general setups. First, the effects of alternative workload distributions between general-purpose processors and accelerators are not well-understood. Second, accelerators have limited capabilities for managing external system resources, such as communication and I/O devices, thus requiring support from general-purpose processors and special consideration while designing the resource management software. Finally, suitable programming models that adapt to the varying capabilities of the accelerator-type components have not been developed, forcing application writers who want to use accelerators on clusters to micro-manage resources.

A. Targeted Environments

Addressing the inherent imbalances of accelerator-based asymmetric clusters while hiding the associated complexity from users is key to achieving high performance and high productivity. Although designing for all possible resource configurations and types of accelerators is very complicated, in this paper, we answer to this challenge by designing and evaluating four alternatives for realizing asymmetric, accelerator-based clusters. We characterize our designs based on the general-purpose computing and system management capabilities of the accelerators. More specifically, we consider three classes of accelerators:

Self-managed well-provisioned accelerators: These accelerators have high compute density, along with on-chip capabilities to efficiently run control code and self-manage I/O and communication. For example, an accelerator coupled with several general-purpose processor cores on the same chip falls into this category. The on-chip computational power of the general-purpose cores and the amount of memory attached to the accelerators is assumed to be sufficient for self-management, in the sense that the control code running for scheduling tasks and performing communication on the general-purpose cores does not become the major performance bottleneck.

* Also with the Department of Computer Science, University of Crete, GR-71490, Heraklion, Greece.

Resource-constrained well-provisioned accelerators: These accelerators have high compute density but insufficient on-chip general-purpose computing capability for running control code and/or insufficient on-board memory for self-managing I/O and communication. I/O and communication are managed by an external, dedicated, compute node with general-purpose cores, which acts as a *driver* for the accelerators.

Resource-constrained shared-driver accelerators: These accelerators are similar to the previous case, however drivers are shared among several accelerators, to yield a potentially more cost-efficient design.

We develop a programming model for the above types of resources, which hides the architectural asymmetry while exploiting the computational density of the accelerators. For this purpose, we adapt and extend MapReduce [18] programming model. Our design uses a dynamic data streaming approach to effectively support MapReduce computations, as well as adaptive resource scheduling that factors in the performance and capabilities of asymmetric components and strives to overlap I/O and communication latencies. Our framework implements data transfers and workload scheduling transparently, while adapting the parameters of data streaming and task scheduling to the application's requirements at runtime, thereby relieving programmers of some significant programming effort. By contrast, previously developed data distribution and task management libraries for asymmetric accelerator-based architectures [19], delegate parameterization of data transfers and workload scheduling to the programmer.

Specifically, this paper makes the following contributions:

- An exploration and evaluation of design alternatives for distributed asymmetric clusters with accelerators;
- An emulation and evaluation of various classes of accelerators, with varying general-purpose computing capabilities;
- An implementation of the MapReduce programming model on asymmetric clusters comprising accelerators, both with and without well-provisioned general-purpose computing resources;
- A thorough performance analysis of our design alternatives in terms of scalability, adaptation to various computation densities, and resource conservation capability.

Our evaluation using representative MapReduce benchmarks (WordCount, Histogram, Linear Regression, K-Means) on an asymmetric cluster with Cell processors serving as accelerators, shows that our approaches significantly improve system performance compared to static, non-streaming schemes for scheduling workload and data transfers. We achieve performance improvement of 62.5%, 73.1%, and 82.2% using accelerators that have limited general-purpose resources, well-provisioned shared general-purpose resources, and well-provisioned dedicated general-purpose resources, respectively. Moreover, our techniques adapt effectively to the relative computation to data transfer density of applications by converging to optimal parameters for data decomposition and streaming at runtime.

II. Background and Motivation

In this section, we discuss the relevant background, related work and motivation for our investigation of various design and resource management alternatives for large-scale accelerator-based asymmetric clusters.

A. Commodity Accelerators

The use of commodity off-the-shelf components in large-scale data centers, e.g., Google [20], Amazon's EC2 [21], etc., is pervasive. Such commoditization is now becoming a norm for asymmetric accelerator-type processors such as the IBM-Sony-Toshiba Cell processor [22], [23] and NVIDIA GPU-based graphics engines [24], [25], consequently, making them cheaper and affordable. This in turn facilitates the building of asymmetric accelerator-based clusters, similar to the framework that we consider in this study.

Accelerators provide much higher performance to cost ratio compared to conventional processors. Thus, a properly designed accelerator-based asymmetric cluster has the potential to provide very high performance at a fraction of the cost and operating budget of a traditional symmetric cluster. Unfortunately, accelerators also pose challenges to programming and resource management. Programming accelerators requires working with multiple ISAs and multiple compilation targets. Accelerators typically have much higher compute density and raw performance than conventional processors, therefore coupling accelerators with conventional processors may introduce imbalance between the two. Accelerators also typically have limited local storage and limited – if any – support for system services such as I/O. To ensure overall high efficiency, resource management on accelerator-based systems needs to orchestrate carefully data transfers and work distribution between heterogeneous components.

We use the Cell processor [26]–[28], on commodity Sony PS3 nodes, in this study. The Cell is a suitable resource that can serve as an accelerator component in high-performance computing (HPC) setups envisioned in this work. The Cell is a heterogeneous chip multi-processor with one general-purpose PowerPC SMT core (the Power Processing Element – PPE), and eight vector-only cores (the Synergistic Processing Elements – SPEs), which are specialized for acceleration of data-parallel computations. The PPE functions as a front-end processor for scheduling work and distributing data between SPEs, as well as for running the operating system. The SPEs have private address spaces and the programmer is responsible for moving data between the main memory and each SPE's local storage using the Cell's coherent asynchronous DMA mechanism. This facility allows the programmer to explicitly manage the

data flow between Cell components, e.g., for improving I/O performance [15]. Finally, in current installations, the PPE runs Linux with Cell-specific extensions that provide user-space libraries access to the PPEs. The Cell has also been used as the compute-engines in the IBM Roadrunner [14], the world's second faster computer, making it an obvious candidate to be used as computational accelerator in this work.

B. MapReduce Programming Model

MapReduce is an emerging programming model for large-scale data processing on clusters and multi-core processors [18], [29]–[31]. Current trends show that the model is widely used as a high-productivity alternative to traditional parallel programming paradigms for a variety of applications, ranging from enterprise computing [21], [32] to peta-scale scientific computing [30], [31], [33]. Several research activities engage in porting MapReduce to multi-core architectures [30], [31], whereas recently, vendors such as Intel begun supporting MapReduce in their libraries and compilers [6], [34].

MapReduce assumes homogeneous components, and any work item can be scheduled on any of the available nodes. Recent work [35] addresses performance heterogeneity for virtualized nodes [21]. However, inherent architecture heterogeneity remains a problem when the cluster components include specialized accelerators, as the mapping function should now consider individual component capabilities and limitations. Furthermore, publicly available implementations of MapReduce, such as Hadoop [32], assumes that data is available on local disks of components and can be accessed fast. Given limited resources of accelerators, this assumption may not hold, thus creating an imbalance between resource-rich control components and resource-constrained compute components.

III. Design

In this section, we discuss the design of accelerator-based asymmetric distributed systems to efficiently support large-scale parallel programming models such as MapReduce [18].

A. Architecture Overview

We opt to use MapReduce due to the simplicity and scalability of the programming model in large-scale data processing computations running on loosely coupled systems. MapReduce better equips us for managing heterogeneous components in our system than a more traditional tightly-coupled approach used in standard clusters.

We arrange our resources as shown in Figure 1. A general purpose well-provisioned multi-core server acts as a dedicated front-end *manager* for the cluster. The server manages a number of back-end accelerator-based nodes and is responsible for scheduling jobs, distributing data, allocating work between compute nodes, and providing other support

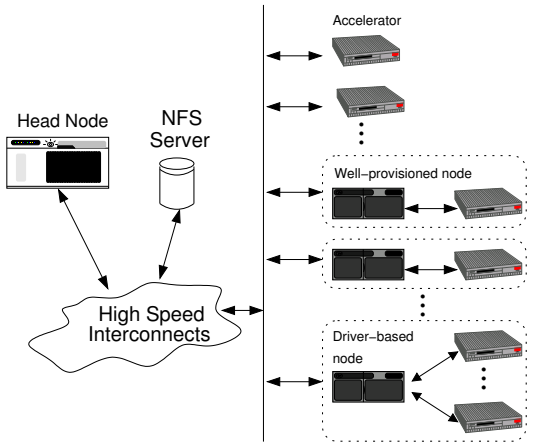
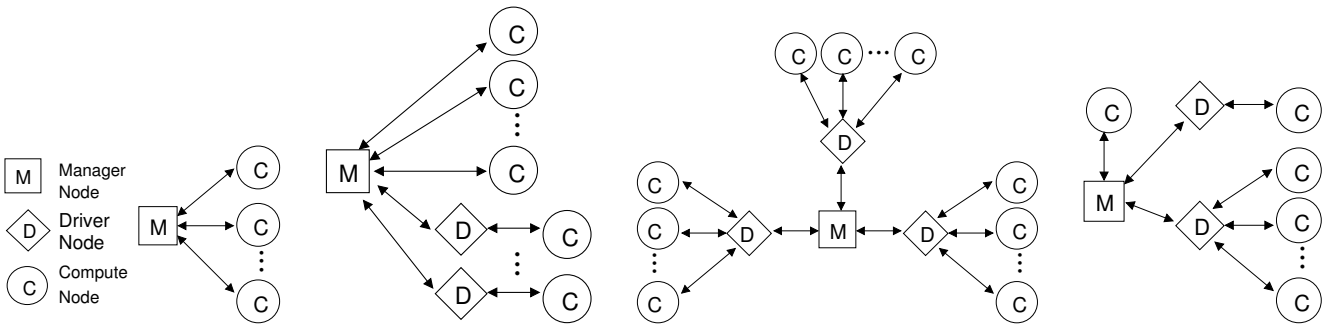


Figure 1. High-level overview of an accelerator-based asymmetric distributed system.

services as the front-end of the cluster. The brunt of processing load is carried by the Cell-based accelerator nodes. The manager divides the MapReduce tasks (map, reduce, and sort etc.) in small workloads, and assign these workloads to the attached accelerator-based nodes. Irrespective of the type of back-end nodes, the manager transparently distributes and schedules the workload to them. If the back-end is a self-managed accelerator, its general-purpose core uses MapReduce to map the assigned workload to the accelerator cores (SPEs). In contrast, if the back-end is driver-based, the driver components further distribute the assigned workload to the attached accelerator node(s). Note that the manager differs from a driver. Drivers execute control tasks for communication and I/O on behalf of accelerators, whereas the manager controls work and data distribution for the entire cluster. This model can be thought of as a hierarchical MapReduce: each level maps the workload to the next level of nodes, until it reaches the compute node, i.e., the Cell processor, where the generic on-chip core maps the workload to the accelerators.

B. Programming Asymmetric Clusters

From an application programmer's point of view, irrespective of the resource configuration employed, MapReduce is used on asymmetric resources as follows. The application is divided into three parts. (i) The code to initialize the runtime environment. This corresponds to the time spent in a MapReduce application but outside of the actual MapReduce work (initialization, intermediate data movement, finalization). This part is unique to our design and does not have a corresponding operation in standard MapReduce. (ii) The code that runs on the accelerator cores and does the actual work of the application. This is similar to a standard MapReduce application running on a small portion of the input data that has been assigned to the compute node. It includes



(a) *Conf I:* Self-managed well-provisioned accelerators.

(b) *Conf II:* Resource-constrained well-provisioned accelerators.

(c) *Conf III:* Resource-constrained shared-driver accelerators.

(d) *Conf IV:* Mixed accelerators.

Figure 2. Resource configurations for enabling asymmetric clusters.

both the map phase to distribute the workload between the accelerator cores, and the reduce phase to merge the data produced from accelerators. (iii) The code to merge partial results from each compute node into a complete result set either by the manager or the driver node. This is called every time a result is received from a compute node and constitutes a Global Merge phase that is identical in operation to the reduce phase on each compute node. The only difference is that the Global Merge on the manager works with entire data sets and produces the final results. All these functions are application-specific and provided by the programmer.

C. Alternate Resource Configurations

We consider four resource configurations for the target asymmetric clusters as shown in Figure 2. The configurations are driven by the type of the back-end components used, as well as by economical constraints and performance goals. In all cases, the manager and all back-end nodes are connected via a high-speed commodity network, e.g., Gigabit Ethernet. Application data is hosted on a distributed file system (NFS [36] in our implementation).

The first configuration (Figure 2(a)) we consider is that of self-managed well-provisioned accelerators (*Conf I*), connected directly to the manager. A blade with Cell processors [37] including multi-Gigabyte DRAM and high-speed network connectivity would fall into this category. Small-scale academic settings may also adopt such a configuration, using, e.g., PS3 nodes and scaling down the workload per PS3 so as to not exceed the limited DRAM capacity and not stress the limited general-purpose processing capabilities of the PS3. The compute nodes execute directly all MapReduce tasks and the manager merges partial results from the computes nodes.

The next configuration (Figure 2(b)) uses resource-constrained well-provisioned accelerators (*Conf II*). Each driver provides large memory space, communication and I/O capabilities to an individual resource-constrained accelerator, e.g. a PS3. The manager distributes input data to the driver nodes in large chunks. The driver nodes proceed by streaming these chunks to the attached accelerators. Accelerators execute the MapReduce tasks, however, partial results produced by accelerators are merged at the corresponding driver nodes and the manager executes the global merge operation on the results received from the driver nodes.

The use of a single driver per resource-constrained accelerator is not always justifiable as one accelerator may not be able to fully utilize the driver's resources. In contrast, a single manager may not be sufficient to match the data demands of many accelerators simultaneously. We address this by using a hierarchical setup (*Conf III*), so that each driver node manages multiple accelerator nodes (Figure 2(c)).

Finally, an asymmetric system may employ a mix of the above configurations based on particular requirements. We capture this mix in our last configuration (*Conf IV*) (Figure 2(d)). In this case, the manager is agnostic of the class of the attached compute nodes and simply divides the input workload between available compute nodes. The execution of MapReduce tasks and merging of partial results are managed automatically at each component, while the final result is produced by the manager, which performs the global merge of the results received from the attached drivers.

D. Addressing Manager-Accelerator I/O Mismatch

The inherent asymmetry between the cluster components may lead to performance degradation, especially due to communication delays associated with data distribution and collection by the manager. Thus, it is critical to handle all communication with different components of the system asynchronously. Asynchronous communication requires careful consideration. If chunks from consecutive input data are distributed to multiple compute nodes, it would require time-consuming complex sorting and ordering to ensure proper merging of the results from individual compute nodes into a consolidated result set. We address this issue by using a separate handler thread on the manager for each of

the compute nodes. Each handler works with a consecutive fixed portion of the data to avoid costly ordering operations. Each handler thread is also responsible for receiving all the results from its associated compute node and performs an application-specific merge operation on the received data. This design leverages multi-core or multi-processor head nodes effectively. Moreover, we use well-established techniques such as double buffering to avoid I/O delays when transferring data between manager/driver and compute nodes. In our experimental testbed, which uses PS3s as accelerators, double buffering combined with the exploitation of multiple cores for the communication with accelerators, overcomes the bandwidth limitations in the network that connects the PS3s with the x86 manager and driver nodes.

The novelty of our design lies in the adoption of a streaming approach to supporting MapReduce. Statically allocating workloads to compute nodes, as is the case in standard MapReduce setups, could saturate the limited non-computation resources on the asymmetric compute nodes and negate any performance benefits. Instead, we slice the input into work units with sizes based on the capabilities of back-end nodes at each level of the cluster hierarchy and stream the slices to the compute nodes, which can then be processed efficiently.

E. Capability-Aware Workload Distribution

Traditional MapReduce designs do not consider individual components capabilities since they assume homogeneous components as compute nodes. Our design, however, has to factor in the capabilities of back-end resources when allocating workloads. For self-managed resources or drivers in other configurations, this task is straightforward. The manager divides the input data and hands it over to the nodes being directly managed. The actual assignment is done by either copying the data to the nodes' local storage or providing them with pointers to the files on the distributed file system. This approach is easy to implement and lightweight for the manager, as the manager does not need to micro-manage data allocation to the accelerators.

However, data handover cannot be used for resource-constrained nodes due to potential limitations such as inability to directly retrieve the data, bottlenecks on the central file system, or lack of sufficient storage and memory for holding local copies. An alternative, that we adopt, is to divide the input data into chunks, with sizes based on the capabilities of compute nodes. Our runtime environment controls the size of these chunks, so that each chunk can be efficiently processed at the compute nodes without overwhelming their resources, e.g., without memory thrashing. Instead of a single division of data, the runtime environment streams chunks (work units) to the compute nodes until all data has been processed. The concern is that such an approach improves performance on the compute nodes at the cost of increasing the load of the manager. The runtime environment balances the load between the manager and the compute nodes, by controlling the resources dedicated to processing communication with each compute node on the manager and continuously adapting the chunk size on the compute nodes.

In addition to addressing I/O diversity, the manager faces different memory and computation pressure depending on the type of back-end resources. For self-managed nodes, the manager is also responsible for merging the results from each of the accelerators repeatedly for the entire input data. This process can be resource consuming. By contrast, for well-provisioned resources, the resource or driver does most of the merging for the accelerators and the manager simply has to perform a global merge. These factors have to be considered when designing asymmetric clusters, taking also into account workload-specific characteristics.

1) Adapting Workload Size: Our framework adaptively matches the workload assigned to a compute node to its capabilities. To this end, we define an optimal workload size to be the largest amount of data assigned per accelerator, which results in minimum execution time for a given application. The intuition is that if a smaller than optimal size is assigned to an accelerator, it would underutilize available resources and would take more iterations at the manager. In contrast, using a larger size would result in increased iteration time due to memory thrashing and resource saturation on accelerators.

The optimal workload size can be determined either statically or dynamically using an auto-tuning heuristic. We adopt an auto-tuning scheme where the driver or manager sends varying size workloads to accelerator nodes at the start of the application and records the completion time corresponding to each size. For each size, the processing rate is calculated as the fraction $(work\ unit\ size)/(execution\ time)$. The size corresponding to the maximum processing rate is selected as the optimal workload size and is employed for the rest of the application's execution time. The same process is repeated at the drivers to find the optimal workload size allocated from each driver to the attached compute nodes.

F. Supporting MapReduce Operations

Once an application begins execution, the associated manager and accelerator software is started on the respective components and the manager initiates MapReduce tasks on the available accelerator nodes. Once assigned, the tasks self-schedule their work by reading data from the distributed file system, processing it, and returning the results back to the manager in a continuous loop. Once the manager receives the results, it merges them to produce the final result set for the application. After a particular MapReduce task has been completed by a self-managed node, the manager assigns another task to that node. This process continues until the entire input data has been processed by the accelerators. The manager handles the driver nodes similarly.

Table I
RESOURCE DISTRIBUTION UNDER DIFFERENT CONFIGURATIONS.

Configuration	# of Drivers	# of PS3s	PS3s per Driver
Conf I	-	8	-
Conf II	8	8	1
Conf III	2	8	4
Conf IV	5	8	4,1

For driver-based resources, each driver loads a portion of input data into its memory, to ensure that sufficient data is readily available for the accelerator nodes. The driver then initiates the required MapReduce tasks on the accelerator nodes and sends the necessary data to the corresponding resource-constrained accelerators. When all the in-memory loaded data has been processed by the accelerators, the driver loads another portion of the input data into memory and the whole process continues until the entire MapReduce task assigned to the particular driver has been completed by the attached resource-constrained accelerators. The driver also merges the result data produced by the accelerators and the merged result-sets are sent back to the manager.

IV. EVALUATION

In this section, we describe our experimental testbed and the benchmarks that we used. We present results that evaluate different design alternatives for realizing asymmetric distributed systems. We evaluate the MapReduce framework that implements various functionalities discussed in Section III as lightweight libraries for each of the hardware platforms in our configurations, i.e., x86 on the manager and PowerPC on the compute nodes, using about 1600 lines of C code. The libraries provide programmers with necessary constructs for using the framework.

A. Experimental Setup

Our testbed consists of eight Sony PS3s, a manager node, and an 8-node x86 multi-core cluster, where each node can serve as a driver. All components are connected via 1 Gbps Ethernet. The manager has two quad-core Intel Xeon 3 GHz processors, 16 GB main memory, 650 GB hard disk, and runs Linux Fedora Core 8. The manager also runs an NFS server. The driver nodes are identical to the manager except that they have 8 GB of main memory. The PS3 is a hypervisor-controlled platform, and has 256 MB of main memory and a 60 GB hard disk. Of the 8 SPEs of the Cell, only 6 SPEs are visible to the programmer [15], [38] in the PS3. Moreover, each PS3 node has a swap space of 512 MB, and runs Linux Fedora Core 7.

Table I shows the distribution of resources that we use for each of the configurations presented in Section III, in addition to the manager node. Note that in *Conf I*, the PS3s are connected directly to the manager, and in *Conf IV*, four PS3s share a driver, while each of the other four has a dedicated driver. Moreover, in all the test configurations, the total number of accelerators is fixed, i.e., 8 PS3s or 48 SPEs and only the resource arrangement is varied. We used a publicly available MapReduce library implementation for Cell [31], to accelerate data mapping, sorting, partitioning and reduction tasks running on individual PS3s.

Table II
EXECUTION TIME (SEC.) ON STAND-ALONE PS3.

Input (MB)	Linear Regression	Word Count	Histogram	K-Means
4	0.34	1.95	1.06	1.66
64	2.88	501.76	45.66	167.93
128	12.56	-	318.66	-
192	21.81	-	394.78	-
256	34.89	-	-	-

B. Methodology

We focus on evaluating our design decisions and deriving clues about what is the best way to utilize a given set of accelerator-based resources for maximizing performance. We use the following well-known MapReduce applications to study the effect of the various design alternatives for the asymmetric cluster. These applications originate from scientific computing environments, including epidemiology, environmental science, image segmentation, and statistical analysis [39]–[41]. More details on these applications can be found in [31].

- *Linear Regression:* This application takes as input a large set of 2D points, and determines a line of best fit for them.
- *Word Count:* This application counts the frequency of each word in a given document. The output is a list of unique words along with their corresponding occurrence counts.
- *Histogram:* This application takes as input a bitmap image and produces the frequency count of each color composition in the image.
- *K-Means:* This application takes a set of points in an N-dimensional space and groups them into a set number of clusters with approximately equal number of points in each cluster.

C. Results

We first examine how the benchmarks behave under our resource configurations discussed in Section III. Then, we evaluate the effectiveness of our design in managing resource-constrained accelerators by adapting workload size and the consequent impact on the manager and drivers. Finally, we examine the scalability of the design.

Table II shows the average execution time for running the four benchmarks on a stand-alone accelerator without using our framework. Note that Linear Regression is the only benchmark that successfully completes for all input sizes. All other benchmarks incur swapping and run out of swap

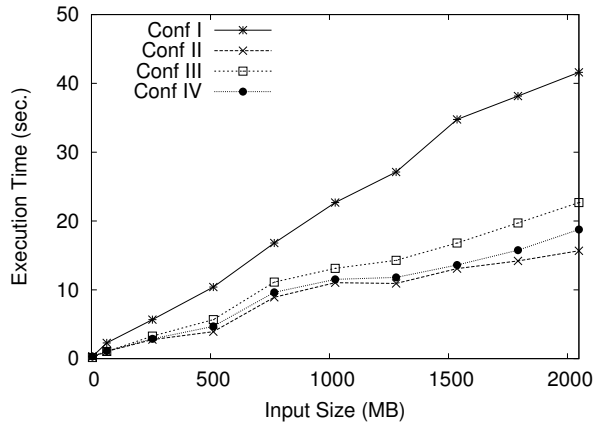


Figure 3. Linear Regression execution time with increasing input size.

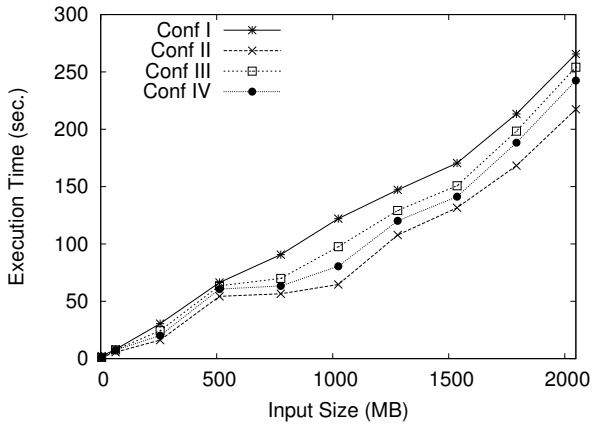


Figure 4. Word Count execution time with increasing input size.

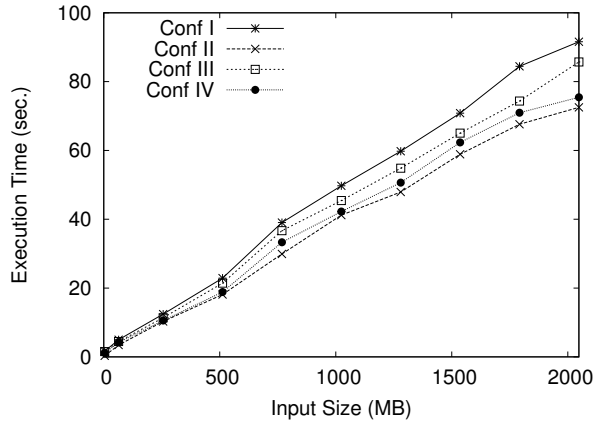


Figure 5. Histogram execution time with increasing input size.

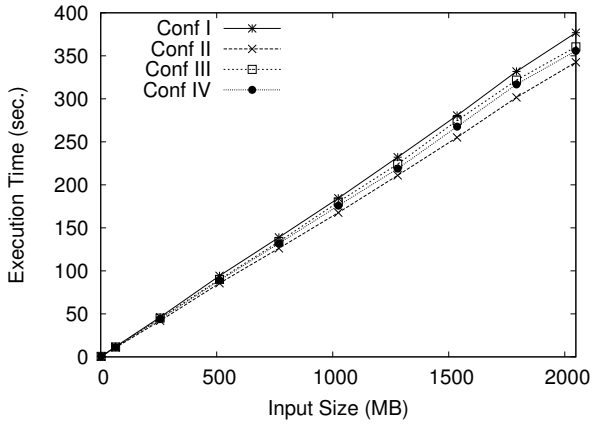


Figure 6. K-Means execution time with increasing input size.

space with smaller input sizes. Also note the rapid growth in the completion time due to excessive swapping as the input size is increased.

1) Benchmark Performance: We first examine the effect of different resource configurations on the average execution time for each of our benchmarks.

Linear Regression: For this benchmark, the input size ranges from 2^{22} points (4 MB) to 2^{31} points (2 GB). Figure 3 shows the average execution time for running the Linear Regression benchmark with increasing input size under different resource configurations. All four resource configurations show similar scaling patterns with the increasing input size. Overall *Conf II* performs 53.1% better than *Conf I*, since each driver node in *Conf II* makes use of its large memory to store and process the intermediate results from the attached PS3s. For similar reasons, i.e., having a higher number of drivers to handle the PS3s, *Conf II* performs 14.4% and 8.0% better than *Conf III* and *Conf IV*, respectively.

Word Count: For Word Count, we observe an exponential growth in memory consumption relative to the input data size, since each word emits additional intermediate data out of the map function. This has the direct impact on the execution time as shown in Table II. For any input size greater than 44 MB, a single accelerator node thrashes and runs out of available swap space (512 MB). However, all the resource configurations in our setup are not only able to process any input size, but also complete the benchmark without thrashing, with linear increase in execution time with increasing input size (Figure 4). Once again, *Conf II* outperforms *Conf I*, *Conf III* and *Conf IV* by 32.5%, 19.2% and 12.7%, respectively, since job scheduling and merging tasks are distributed efficiently between driver nodes.

Histogram: Figure 5 shows the average execution time for running the Histogram benchmark under the four test configurations. On average *Conf II* performs 25.1%, 17.8% and 11.1% better than *Conf I*, *Conf III* and *Conf IV*, respectively. In our experiment with a stand-alone PS3, we observe that the execution time for 192 MB input size is 394.8 seconds because of excessive swapping of intermediate data. This benchmark also shows that our design scales linearly for any input size for all tested configurations.

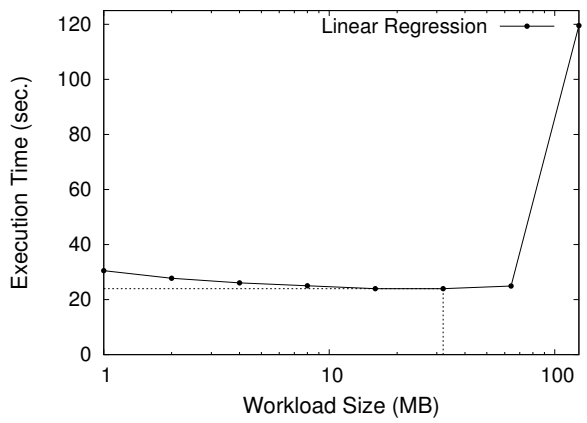


Figure 7. Effect of workload size on execution time.

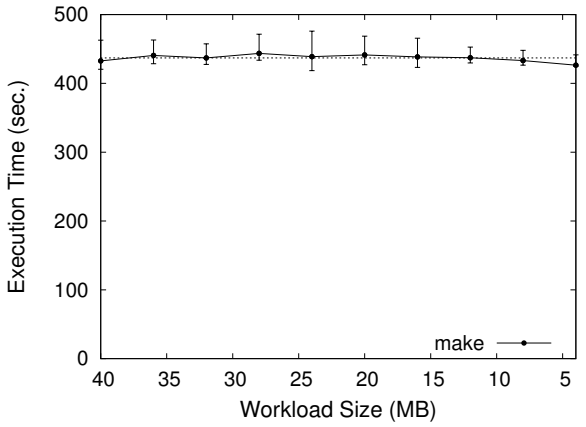


Figure 8. Impact of workload size on the driver.

K-Means: Figure 6 shoes the results for the K-Means benchmark. K-Means uses a different number of iterations for different input sizes. Therefore, considering total execution times for different inputs does not provide a fair comparison of the effect of increasing input size. We remedy this by reporting the execution time per iteration in the figure. The result for this benchmark shows that *Conf II* outperforms *Conf I*, *Conf III* and *Conf IV* by 9.0%, 6.6% and 4.4% respectively. Just as in the case of previous benchmarks, the improvement comes from the fact that in *Conf II* each accelerator node has more memory and computation resources in the form of a dedicated front-end node attached with it.

In summary, memory constraints non-withstanding, all resource configurations using our framework show similar patterns with increasing input size. They also exhibit better memory utilization and enable efficient handling of large data sets. Overall, we observe that *Conf II* gives the best performance across applications, however, it is not cost-effective given the need for dedicated drivers. However, if the number of accelerators per driver increases, accelerators may stress the drivers with management tasks and overwhelm their resources, leading to reduced performance, as we observe for *Conf I* or *Conf III*. The number of accelerators per driver that keeps the workload in limits, is a function of the driver and accelerator capabilities and is different for different resources. Thus, *Conf III* and *Conf IV* offer better choices: they economize on the number of drivers, yet provide performance comparable to *Conf II*.

2) Driving Resource-Constrained Nodes: We focus on *Conf I* to examine the performance of our design in handling resource-constrained nodes. For this discussion, we use manager and driver interchangeably as the role of manager in *Conf I* is identical to the role of driver in other configurations.

Adapting Workload: Varying workload size affects the processing time on a node. To show this, we use one PS3 connected to the manager, and run Linear Regression with an input size of 512 MB. Figure 7 shows that as the workload increases, the execution time first decreases to a minimum and eventually increases exponentially. The valley point (shown by dashed line) indicates the size after which the compute node starts to page. Notice that the curve is almost flat before the valley indicating no extra overhead for processing more data. Also, using a smaller work unit size increases the manager's load, as the manager now has to handle a larger number of chunks for a given input size. We argue that using the valley point as the workload size provides the best trade-off between node performance and manager performance.

Next, we evaluate our framework's ability to dynamically determine the optimal workload. We follow an experimental process to discover optimal workload size. We manually determine the maximum workload for each application that can run on a single PS3 without paging, and compare it with the work unit size that *Conf I* determines at runtime. Table III shows the result. Our framework is able to dynamically determine an appropriate workload that is close to the one found manually and this determination on average across our benchmarks takes under 0.93 seconds. This is negligible, i.e., less than 0.5% of the total application execution times when the input size is 2 GB. Thus, adaptive workload determination in our framework is efficient as well as reasonably accurate.

Impact on the Driver: We determine the effect of varying workload sizes on driver performance. We use *Conf I*, however, with a driver node instead of the manager. First, we start a long running job (Linear Regression) on the driver node. Next, we determine the time it takes to compile a large project (Linux kernel 2.6) on the driver, while the MapReduce task is running. We repeat the steps as we decrease the workload size, potentially increasing the processing requirements from the driver. We repeat the experiment 10 times and record the minimum, maximum,

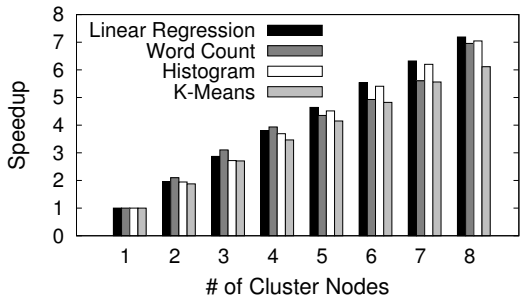
Table III
PERFORMANCE OF ADAPTING WORKLOAD TO ACCELERATORS.

Application	Hand-Tuned Size (MB)	Our Framework		
		Size (MB)	# Iterations	Time (s)
Linear Regression	32	30	16	0.65
Word Count	3	2	8	1.82
Histogram	2	1	4	0.15
K-Means	0.37	0.12	16	1.09

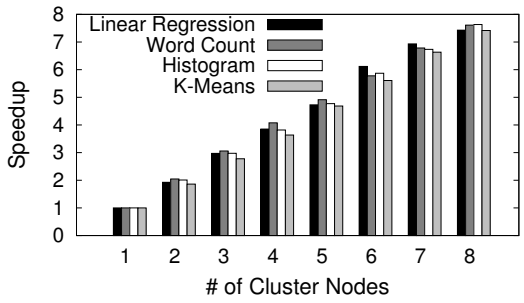
and average time for the compilation as shown in Figure 8. The horizontal dashed line in the figure shows the overall average of compile time across all studied workload sizes. Given that the overall average remains within the minimum and maximum times, we can infer that the variations in the compile time curve are within the margin of error. Thus, the relatively flat curve indicates that our framework has a constant load on the driver and can support various workloads without the driver becoming a bottleneck.

3) Scaling Characteristics: We observe how the performance of our benchmarks scale with the number of accelerator nodes using *Conf I*, *Conf II*, and *Conf III*. Figure 9(a) shows the speedup in performance normalized to the case of 1 node in *Conf I* and *Conf II*. Both these configurations have similar speedups because of similar manager to compute node relationship, and are shown in a single graph. For *Conf III*, we only have enough PS3s to scale up to using four drivers with four PS3s each. However, we emulate up to 8 drivers as follows. During our tests with 1 to 4 drivers, we observe near identical load on the manager from each of the drivers. Based on this observation, we create a test-loader that generates the same requests to the manager as that of a driver with accelerators and use it to scale the experiment beyond four accelerators. Figure 9(b) shows the speedup for our benchmarks in *Conf III*. We use the same input size for all runs of an application. However, the input sizes for the different applications are chosen to be large enough to benefit from using 8 nodes: 512 MB for Linear Regression and Histogram, 200 MB for Word Count, and 128 MB for K-Means. The curve of K-Means is based on time per iteration, as explained earlier.

Although we are only able to evaluate scaling on the relatively modest scale of 8 nodes, our results show that our framework scales almost linearly as the number of compute nodes increases and this behavior persists for all the benchmark. However, we observe that the improvement trend does not hold for all benchmarks in *Conf I/II* when the eighth node is added. Upon further investigation, we find that the network bandwidth utilization for such cases is quite high, as much as 107 MB/s compared to the maximum observed value of 111 MB/s on our network, measured using remote copy of a large file. High network utilization introduces communication delays even with double buffering and prevents our framework from achieving a linear speedup.



(a) *Conf I/II.*



(b) *Conf III.*

Figure 9. Effect of scaling on resource configurations.

However, if the ratio of time spent in computation compared to that in communication is high, which is the case in scientific applications, we can obtain near linear speedup. We test this hypothesis by artificially increasing our compute time for Linear Regression by a factor of 10, which results in a speedup of 7.8. For *Conf III*, no such network bottleneck exists, since each driver manages the attached accelerator using a dedicated connection.

V. CONCLUSION

This paper presents four design alternatives and configurations for building asymmetric clusters with PS3 accelerators at the compute nodes and multi-core x86 servers at driver and manager nodes. We presented the design, implementation, and evaluation of different resource configurations by emulating accelerator nodes with varying general-purpose computing capabilities and their impact on overall system performance. We explored this design space by designing an extended MapReduce model for asymmetric HPC clusters, which adopts a data streaming approach to make the data available to the accelerator in a timely fashion. Our implementation of MapReduce hides the imbalance and architectural asymmetry between the general-purpose nodes and accelerator components and uses adaptive resource scheduling by considering the performance and capacities of the components. Thus, our design enables higher performance and better utilization of the available asymmetric components, which in turn helps capacity planning for emerging asymmetric distributed systems. During our

evaluation, we observed a large variance in the performance of different design choices for the asymmetric clusters, which shows that asymmetric clusters are highly sensitive to the design configurations of their general-purpose and accelerator resources.

In our ongoing work, we aim to deploy the framework presented in this paper to non-MapReduce programming models. Moreover, our long-term objective is to develop planning tools and models that will allow system designers to create performance-budget balanced configurations.

ACKNOWLEDGMENT

This research is supported by NSF (grants CCF-0746832, CCF-0346867, CCF-0715051, CNS-0521381, CNS-0720673, CNS-0709025, CNS-0720750), DOE (grants DE-FG02-06ER25751, DE-FG02-05ER25689), IBM through an IBM Faculty Award (grant VTF-874197), and the European Commission (grants MCF-IRG-224759, IST-004408, IST-217068). M. Mustafa Rafique is supported through a Fulbright scholarship.

REFERENCES

[1] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The Impact of Performance Asymmetry in Emerging Multicore Architectures," in *Proc. ISCA.*, 2005.

[2] M. Hill and M. Marty, "Amdahl's Law in the Multi-core Era," Department of Computer Sciences, University of Wisconsin-Madison, Tech. Rep. 1593, Mar. 2007.

[3] M. Pericàs, A. Cristal, F. Cazorla, R. González, D. Jiménez, and M. Valero, "A Flexible Heterogeneous Multi-core Architecture," in *Proc. PACT.*, 2007.

[4] K. R., K. Farkas, N. Jouppi, P. Ranganathan, and D. M. Tullsen, "Processor Power Reduction via Single-ISA Heterogeneous Multi-core Architectures," *Computer Architecture Letters*, vol. 2, 2003.

[5] K. R., D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance," in *Proc. ISCA.*, 2004.

[6] H. Wong, A. Bracy, E. Schuchman, T. Aamodt, J. Collins, P. Wang, G. Chinya, A. Groen, H. Jiang, and H. Wang, "Pangaea: A Tightly-Coupled IA32 Heterogeneous Chip Multiprocessor," in *Proc. PACT.*, 2008.

[7] AMD, "The Industry-Changing Impact of Accelerated Computing," 2008.

[8] D. Bader and V. Agarwal, "FFTC: Fastest Fourier Transform for the IBM Cell Broadband Engine," in *Proc. HiPC.*, 2007.

[9] F. Blagojevic, A. Stamatakis, C. Antonopoulos, and D. Nikolopoulos, "RAxML-CELL: Parallel Phylogenetic Tree Construction on the Cell Broadband Engine," in *Proc. IPDPS.*, 2007.

[10] G. Buehrer and S. Parthasarathy, "The Potential of the Cell Broadband Engine for Data Mining," Department of Computer Science and Engineering, Ohio State University, Tech. Rep. TR-2007-22, 2007.

[11] B. Gedik, R. Bordawekar, and P. S. Yu, "Cellsort: High performance sorting on the cell processor," in *Proc. VLDB.*, 2007.

[12] S. Heman, N. Nes, M. Zukowski, and P. Boncz, "Vectorized Data Processing on the Cell Broadband Engine," in *Proc. DaMoN.*, 2007.

[13] F. Petrini, G. Fossum, J. Fernández, A. L. Varbanescu, M. Kistler, and M. Perrone, "Multicore surprises: Lessons learned from optimizing sweep3d on the cell broadband engine," in *Proc. IPDPS.*, 2007.

[14] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: The architecture and performance of Roadrunner," in *Proc. SC.*, 2008.

[15] M. M. Rafique, A. R. Butt, and D. S. Nikolopoulos, "DMA-based Prefetching for I/O-intensive Workloads on the Cell Architecture," in *Proc. CF.*, 2008.

[16] ClearSpeed Technology, *ClearSpeed whitepaper: CSX processor architecture*, 2007. [Online]. Available: http://www.clearspeed.com/docs/resources/ClearSpeed_Architecture_Whitepaper_Feb07v2.pdf

[17] Jason Cross, "A Dramatic Leap ForwardGeForce 8800 GT," Oct 2007, http://www.extremetech.com/article2/0,1697,2209197,00.asp.

[18] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. USENIX OSDI.*, 2004.

[19] C. H. Crawford, P. Henning, M. Kistler, and C. Wright, "Accelerating computing with the cell broadband engine processor," in *Proc. CF.*, 2008.

[20] L. A. Barroso, J. Dean, and U. Hlzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[21] Amazon, "Amazon Elastic Compute Cloud (Amazon EC2)," http://www.amazon.com/b?ie=UTF8&node=201590011.

[22] "Astrophysicist Replaces Supercomputer with Eight PlayStation 3s," http://www.wired.com/techbiz/it/news/2007/10/ps3_supercomputer.

[23] Mueller, "NC State Engineer Creates First Academic Playstation 3 Computing Cluster," http://moss.csc.ncsu.edu/~mueller/cluster/ps3/coe.html.

[24] GraphStream, Inc., "GraphStream scalable computing platform (SCP)," 2006, http://www.graphstream.com.

[25] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. H. M. Buijssen, M. Grajewski, and S. Turek, "Exploring weak scalability for FEM calculations on a GPU-enhanced cluster," *Parallel Computing.*, vol. 33, no. 10-11, pp. 685–699, 2007.

[26] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata, "Cell Broadband Engine Architecture and its first implementation - A performance view," *IBM J. Res. Dev.*, vol. 51, no. 5, pp. 559–572, 2007.

[27] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, D. Shippy, "Introduction to the Cell multiprocessor," *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 589–604, 2005.

[28] IBM Corp., "Cell Broadband Engine Architecture (Version 1.02)," 2007.

[29] B. He, W. Fang, Q. Luo, N. Govindaraju, and T. Wang, "Mars: A MapReduce Framework on Graphics Processors," in *Proc. PACT.*, 2008.

[30] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating MapReduce for Multi-core and Multiprocessor Systems," in *Proc. HPCA.*, 2007.

[31] M. de Kruijf and K. Sankaralingam, "MapReduce for the Cell B.E. Architecture," Department of Computer Sciences, The University of Wisconsin-Madison, Madison, WI, Tech. Rep. TR1625, 2007.

[32] Apache Software Foundation., "Hadoop," May 2007, http://hadoop.apache.org/core/.

[33] Adam Pisoni, "Skynet," Apr. 2008, http://skynet.rubyforge.org.

[34] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: A Programming Model for Heterogeneous Multi-core Systems," in *Proc. ASPLOS.*, 2008.

[35] M. Zaharia, A. Konwinski, and A. D. Joseph, "Improving mapreduce performance in heterogeneous environments," in *Proc. USENIX OSDI.*, 2008.

[36] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network file system," in *Proc. Summer USENIX*, 1985.

[37] A. K. Nanda, J. R. Moulic, R. E. Hanson, G. Goldrian, M. N. Day, B. D. D'Arnora, and S. Kesavarapu, "Cell/B.E. blades: building blocks for scalable, real-time, interactive, and digital media servers," *IBM J. Res. Dev.*, vol. 51, no. 5, pp. 573–582, 2007.

[38] J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra, "The PlayStation 3 for High-Performance Scientific Computing," *Computing in Science and Engineering*, vol. 10, no. 3, pp. 84–87, 2008.

[39] P. Burton, L. Gurrin, and P. Sly, "Extending the simple linear regression model to account for correlated responses: An introduction to generalized estimating equations and multi-level mixed modelling," *Statistics in Medicine*, vol. 17, no. 11, pp. 1261–1291, 1998.

[40] A. Guisan, T. C. Edwards, and T. Hastie, "Generalized linear and generalized additive models in studies of species distributions: setting the scene," *Ecological Modelling*, vol. 157, no. 2-3, pp. 89 – 100, 2002.

[41] T. N. Pappas and N. S. Jayant, "An adaptive clustering algorithm for image segmentation," *IEEE Transactions on Signal Processing*, vol. 40, no. 4, pp. 901–914, 1992.