2012 IEEE 4th International Conference on Cloud Computing Technology and Science

A Broadband Embedded Computing System for MapReduce Utilizing Hadoop

YoungHoon Jung Richard Neill Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, NY, 10027
Email: {jung,rich,luca}@cs.columbia.edu

***Abstract*—An expanding wealth of ubiquitous, heterogeneous, and interconnected embedded devices is behind most of the exponential growth of the "Big Data" phenomenon. Meanwhile, the same embedded devices continue to improve in terms of computational capabilities, thus closing the gap with more traditional computers. Motivated by these trends, we developed a heterogeneous computing system for MapReduce applications that couples cloud computing with distributed embedded computing. Specifically, our system combines a central cluster of Linux servers with a broadband network of embedded set-top box (STB) devices. The MapReduce platform is based on the Hadoop software framework, which we modified and optimized for execution on the STBs. Experimental results confirm that this type of heterogeneous computing system can offer a scalable and energy-efficient platform for the processing of large-scale data-intensive applications.**

***Keywords*-MapReduce; Hadoop; Embedded Computing; Benchmark; Set-Top Box (STB); Broadband Network.**

I. Introduction

The growth in the amount of data created, distributed and consumed continues to expand at exponential rates: according to a recent research report from the International Data Corporation, the amount of digital information created and replicated has exceeded the zettabyte barrier in 2010 and this trend is expected to continue to grow "as more and more embedded systems pump their bits into the digital cosmos" [8]. In recent years the MapReduce framework has emerged as one of the most widely used parallel computing platforms for processing data on very large scales [17]. While MapReduce was originally developed at Google [9], open-source implementations such as Hadoop [2] are now gaining widespread acceptance.

The ability to manage and process data-intensive applications using MapReduce systems such as Hadoop has spurred research in server technologies and new forms of Cloud services such as those available from Yahoo, Google, and Amazon.

Meanwhile, the Information Technology industry is experiencing two major trends. On one hand, computation is moving away from traditional desktop and department-level computer centers towards an infrastructural core that consists of many large and distributed data centers with high-performance computer servers and data storage devices, virtualized and available as Cloud services. These large-scale centers provide all sorts of computational services to a multiplicity of peripheral clients, through various interconnection networks. On the other hand, the increasing majority of these clients consist of a growing variety of embedded devices, such as smart phones, tablet computers and television set-top boxes (STB), whose capabilities continue to improve while also providing data locality associated to data-intensive application processing of interest [21], [22]. Indeed, the massive scale of today's data creation explosion is closely aligned to the distributed computational resources of the expanding universe of distributed embedded systems and devices. Multiple Service Operators (MSOs), such as cable providers, are an example of companies that drive both the rapid growth and evolution of large-scale computational systems, consumer and business data, as well as the deployment of an increasing number of increasingly-powerful embedded processors.

Our work is motivated precisely by the idea that the ubiquitous adoption of embedded devices by consumers and the combination of the technology trends in embedded systems, data centers, and broadband networks open the way to a new class of heterogeneous Cloud computing for processing data-intensive applications. In particular, we propose a *broadband embedded computing system for MapReduce utilizing Hadoop* as an example of such systems. Its potential application domains include: ubiquitous social networking computing, large-scale data mining and analytics, and even some types of high-performance computing for scientific data analysis. We present a heterogeneous distributed system architecture which combines a traditional cluster of Linux blade servers with a cluster of embedded processors interconnected through a broadband network to offer massive MapReduce data-intensive processing potential (and, potentially, energy and cost efficiency).

Contributions. We have implemented a prototype small-scale version of our proposed system where a *Linux Cluster* features nine high-end blade servers and an *Embedded Cluster* consists of a network of 64 STBs. The two clusters are interconnected through the broadband network of a complete head-end cable system, as described in Section II. While the cable system remains fully operational in terms of its original function (e.g. by distributing streaming-video

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

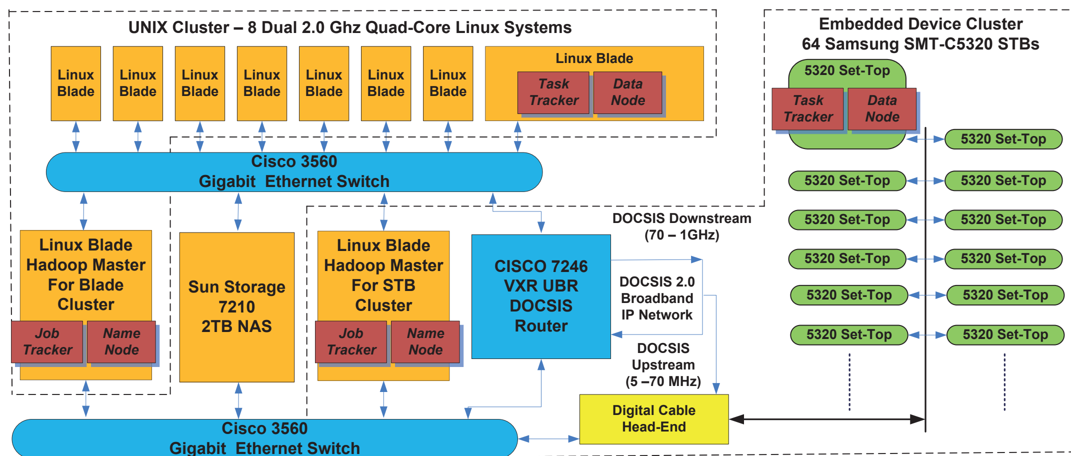


Figure 1. Architecture of the broadband embedded computing system for MapReduce utilizing Hadoop.

content to the STBs which render it to their displays), it is possible to simultaneously and effectively execute other MapReduce applications by leveraging the additional computation resources that are available in the STB multi-core processors.

Specifically, we ported the Hadoop MapReduce framework to our broadband embedded computing system. As discussed in Section III, this porting posed important challenges in terms of software portability and resource management. We addressed these challenges in two ways. First, we developed porting techniques for embedded devices that leverages back-porting of enterprise software in order to implement the Hadoop system for embedded environments. Second, to execute MapReduce applications on such resource-constrained embedded devices as STBs, we optimized both memory and storage requirements by eliminating unnecessary software components of the Hadoop platform. The result is an embedded version of the Hadoop framework.

In Section IV we present a set of experiments which confirm that our embedded system implementation of the Hadoop runtime environment and related software libraries runs successfully a variety of MapReduce benchmark applications. Also, in order to gain further insight into the relative performance scaling of the Embedded Cluster versus the Linux Cluster while running MapReduce applications, we varied the number of processing elements (which correspond to the number of *Hadoop nodes*) and the size of the input data. Overall, the experimental results expose the Embedded Cluster performance sensitivity to certain classes of MapReduce applications and indicate avenues of future research to improve our system.

II. The System Architecture

Fig. 1 provides an overview of the architecture of the system that we developed and built: this is a heterogeneous system that leverages a broadband network of embedded devices to execute MapReduce applications by utilizing Hadoop. It is composed of four main subsystems.

Linux Blade Cluster. The Linux Cluster consists of a traditional network of nine blade servers and a Network Attached Storage (NAS). Each blade has two quad-core 2GHz Xeon processors running Debian Linux with 32GB of memory and a 1Gb/s Ethernet interface. One of the nine blades is the Hadoop master host acting both as NameNode and JobTracker for the MapReduce runtime management [2]. Each of the other eight blades is a Hadoop slave node, acting both as DataNode and TaskTracker [2] while leveraging the combined computational power of the eight processing cores integrated on the blade. The blades use the Network File System (NFS) to mount the 2TB Sun storage array which provides a remote common file-system partition to store applications for each of the executing Hadoop MapReduce applications. For storing the Hadoop Distributed File System (HDFS) data, the blades use their own local hard-disk drive (HDD).

Embedded STB Cluster. The Embedded Cluster consists of 64 Samsung SMT-C5320 set-top boxes (STB) that are connected with a radiofrequency (RF) network for data delivery using MPEG and DOCSIS transport mechanisms. The Samsung SMT-C5320 is an advanced (2010-generation) STB featuring an SoC with a Broadcom MIPS 4000 class processor, a floating-point unit, dedicated video and 2D/3D-graphics processors with OpenGL support, 256MB of system memory, 64MB internal Flash memory, 32GB of external Flash memory accessible through USB, and many network transport interfaces (DOCSIS 2.0, MPEG-2/4 and Ethernet). Indeed, an important architectural feature of modern STBs is the heterogeneous multi-core architecture design which allows the 400MHz MIPS processor, graphics/video processors, and network processors to operate in parallel over independent buses. Hence, user-interface applications (such as the electronic programming guides) can execute in parallel with any real-time video processing. From the viewpoint of running Hadoop applications as a slave node, however, each STB can leverage only the MIPS processor

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

while acting both as DataNode and TaskTracker. [1] This is an important difference between the Embedded Cluster and the Linux Cluster. Finally, in each STB, a 32GB USB memory stick is used for HDFS data storage, while NFS is used for Java class storage.

Network. The system network is a managed dedicated broadband network which is divided into three IP subnets to isolate the traffic between the DOCSIS-based broadband Embedded Cluster network, the Linux Cluster network, and the digital cable head-end. Its implementation is based on two Cisco 3560 1Gb/s Ethernet switches and one Cisco 7246 DOCSIS broadband router. The upper switch in Fig. 1 interconnects the eight blades along with the NAS and master host. The lower switch aggregates all the components on the head-end subnetwork. The DOCSIS subnetwork is utilized by the Embedded Cluster whose traffic exists on both the Linux Cluster and the digital head-end network. The broadband router has 1Gb/s interfaces for interconnection to the Linux Cluster and head-end networks as well as a broadband interface for converting between the DOCSIS network and the Ethernet backbone. Each broadband router can support over 16,000 STBs, thus providing large-scale fan-out from the Linux Cluster to the Embedded Cluster.

Embedded Middleware Stack. The embedded middleware stack is based on Tru2way, a standard platform deployed by major cable operators in U.S. as part of the Open Cable Application Platform (OCAP) developed in conjunction with Cablelabs [4]. Various services are delivered through the Tru2way platform including: chat, e-mails, electronic games, video on-demand (VOD), home shopping, interactive program guides, stock tickers, and, most importantly, web browsing [5]. To enable cable operators and other third-party developers to provide portable services, Tru2way includes middleware based on Java technology that is integrated into digital video recorders, STBs, TVs, and other media-related devices.

Tru2way is based on Java ME (Java Micro Edition) with CDC (Connected Device Configuration) designed for mobile and other embedded devices. The Tru2way standard follows FP (Foundation Profile) and PBP (Personal Basis Profile) including: io, lang, net, security, text, and util packages as well as awt, beans, and rmi packages, respectively. Additional packages include JavaTV for Xlet applications, JMF (Java Media Framework), which adds audio, video, and other time-based media functionalities, and MHP (Multimedia Home Platform), which comprises classes for interactive digital television applications. On top of these profiles, the OCAP API provides applications with Tru2way-specific classes related to hardware, media, and user-interface packages unique to cable-based broadband content-delivery systems.

[1]In the Embedded Cluster, there is also a Linux blade which is the Hadoop master node, acting both as NameNode and JobTracker.

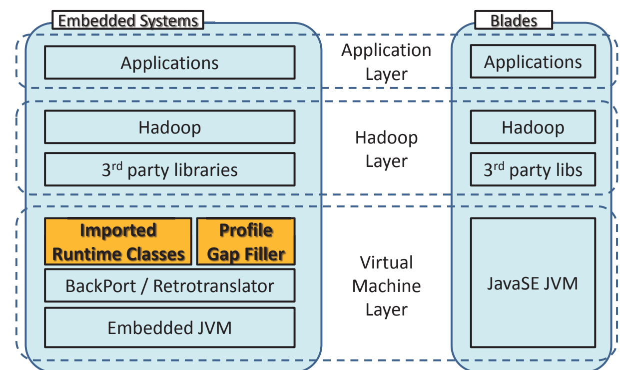


Figure 2. Two software stacks to support Hadoop: STB vs. Linux Blade.

Remark. While this rich set of Java profiles offer additional features to the embedded Java applications, there exists a significant gap between the Java stack provided by Tru2way and the Java Platform Standard Edition (Java SE), which is common to enterprise-class application development. Hence, since the standard Hadoop execution depends on the Java SE environment, we had to develop a new implementation of Hadoop specialized for the embedded software environment that characterizes devices such as STBs. We describe our effort in the next section.

III. Porting Hadoop to the Broadband Embedded System

There are several issues that need to be addressed in order to successfully run Hadoop on a distributed embedded systems like our broadband network of STB devices.

First, Hadoop and Hadoop third-party libraries require many bootstrap classes not supported by the Tru2way JVM. Also, for many classes the Tru2way JVM supports only a subset of methods: e.g., both Tru2way and Java SE have the `java.lang.System` class, but the `java.lang.System.getenv()` method exists only in Java SE.

Second, the Tru2way JVM only supports older versions of Java class file formats while Hadoop is developed using many Java 1.6 language features including: generics, enums, for-each loops, annotations, and variable arguments.

Third, the task of porting Java applications to another JVM with different profiles is quite challenging and, differently from porting native codes to JVM [7], [16], it has not been actively studied in the literature. If not an impossible task, to modify Hadoop and the Hadoop third-party libraries at the source code level is not really practical because there are more than fifty of such libraries and, in some cases, their source code is not available.

Finally, despite all the efforts to improve the JVM portability [23], [24], to port the Java SE JVM to the STB environment is difficult because these embedded devices do not support key features such as frame buffer or native implementations.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

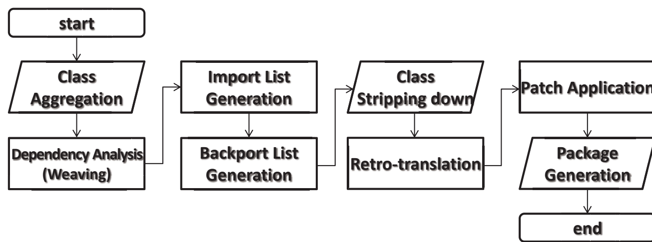


Figure 3. Porting the Hadoop-supporting Java classes to the STB devices.

To address these challenges, we have developed a binary level porting method for embedded devices that imports missing class files and retro-translates all the class files so that the embedded Tru2way JVM can execute them. Our method leverages the Java Backport package, which is the implementation of JSR 166 (`java.util.concurrent` APIs), introduced in Java SE 5.0 and further refined in Java SE 6.0, for older versions of Java platforms [1]. The Retrotranslator has two main functionalities: 1) it translates newer class files into an older format for an older JVM; and, 2) it extends the Backport package so that most Java SE 5.0 features are available for an application that runs on the Java SE 1.4 and Java SE 1.3 JVMs [3]. The runtime classes from those two packages can be added to the Tru2way JVM.

Fig. 2 shows the resulting software stack to support the execution of Hadoop in the embedded environment of an STB running the Tru2way JVM and contrasts it with the traditional software stack based on the Java SE JVM running on a common Linux blade. In particular, the embedded software stack includes the *Imported Runtime Classes*, which are the results of the backporting technique, and the *Profile Gap Filler*, which collects all additional components that were developed specifically for the embedded STB devices.

Fig. 3 illustrates the procedure that we developed to port Hadoop and all the Java packages necessary for running Hadoop to the STB devices. While it was developed and tested for our broadband embedded system, for the most part this procedure is a contribution of general applicability to port Java applications originally developed for the Java SE JVM to other embedded systems which have different and more limited JVMs: e.g., this procedure can be followed also for porting any Java applications to other JVM such as BD-J or Android's Dalvik [13], [18]. The procedure consists of a sequence of eight main steps:

1) Class Aggregation. Here all the input classes are simply copied into a single directory and the priorities among the duplicated or collided classes are determined.

2) Dependency Analysis. For this step, which is key to implementing efficiently a large Java application like Hadoop on resource-constrained embedded devices, we developed a novel dependency analysis technique called *Class Weaving*. This starts by analyzing the class dependencies within a Java package as well as across the packages and then changes the dependency to reuse as much as possible those classes which are available in the embedded Java ME environment. The goal is to generate all the information on class dependencies that is necessary at later steps to minimize the number of classes which will be imported from the various open-source Java SE runtime libraries (and to strip out all unnecessary classes from the original packages.) Fig. 4 illustrates how Class Weaving works: a class dependency tree is generated by analyzing each class while minimizing the number of classes to be imported. For example, Hadoop's `TaskTracker` class uses the `Pattern` class, which in turn uses the `Matcher` class: both these classes exist in Java SE but not in the STB Java ME environment and, therefore, need to be imported. On the other hand, the `Pattern` class uses the `Character` class, which exists also in Java ME and, therefore, it will not be imported from Java SE: instead, the `Pattern` class will be woven to use Java ME's `Character` class.

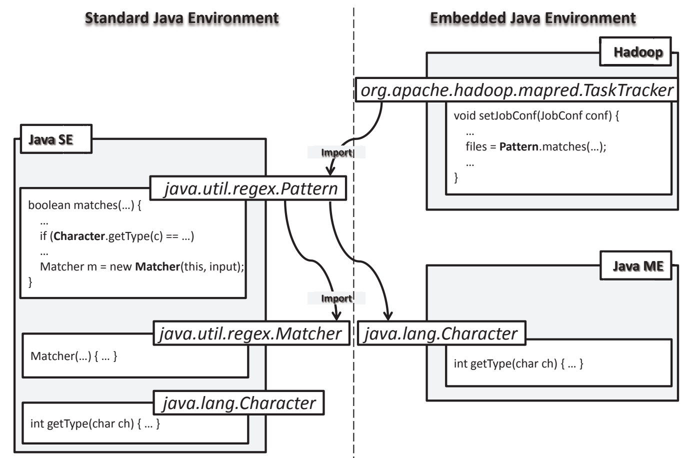


Figure 4. Example of applying the proposed Class Weaving method.

3) Import List Generation. Based on the information collected at the previous step, the list of classes to be imported is generated. At this step, the list can be refined through additional customizations. Unlike most JVMs, some embedded JVMs have their bootstrap classes embedded in a way that are not accessible to the application developers and provide only stub classes to them. For instance, packages like `xerces` or `log4j` do exist in the actual bootstrap classes for internal purposes but are not included in the stub classes.

4) Backport List Generation. The Java class loaders check if the package name of the target class begins with the `'java.'` prefix when the class file location is not in the bootstrap classpaths and, if so, returns an error. To avoid this, the prefix needs to be changed: e.g., in the case of our system with the `'edu.columbia.cs.sld.backport.ocap.java.'` prefix. A list of the mappings between the original and the new prefix is generated for all the imported classes with package names that begin with `'java.'` to be used later in the retro-translation step.

5) Class Stripping Optimization. Since many embedded systems have limited memory and storage resources, only

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

	Hadoop & 3rd-party libs	JavaSE Bootstrap
Before	14490	10110
After	4141	5978

Table 1. Class count before & after class stripping optimization.

the necessary Java classes should be stored in the embedded device. This is achieved by collecting dependency trees that begin with the *seed classes*, which include the entry point `Xlet` class that launches Hadoop DataNode and TaskTracker, various classes that are dynamically loaded from configuration files or from the source code, and the patched classes. In our case, this step results in a 60% reduction of the number of classes that must be deployed in the STBs, as shown in Table 1.

6) Retro-translation. Since the Tru2way JVM recognizes classes up to Major Version Number 48, all the class files with Major Version Number 49 or higher need to be retro-translated. Most packages, including Hadoop, provide classes with major version number 50, which corresponds to Java 1.6. At the binary level, the class file formats and package names of Hadoop, Java SE, and the application libraries need to be properly modified.

7) Patch Application. While a number of classes were imported from open-source Java SE runtime libraries through the Class Weaving technique described above, we had to newly develop a number of missing classes and methods which needed to be optimized before being added to the Java stack of the STBs. The same was necessary for classes that could not be imported from the open-source Java SE runtime library due to the native implementations. Also, patches were necessary to fix some defects found in the Tru2way implementations.

8) Package Generation. This final step generates the packages that will be launched on the Tru2way JVM from the stripped classes, links a custom class loader that will loads user-defined `Mapper` and `Reducer` classes, and binds an entry point `Xlet` that will execute Hadoop DataNode and TaskTracker.

A. Challenges in Porting Hadoop to STB Devices

The number of JVM processes supported in the system is one of the biggest differences between the STB Java environment and a Linux blade server utilizing Java SE. While the users of the latter can launch multiple instances of JVM, only one JVM instance can be launched during boot time within an STB. On the other hand, there are two important behaviors in Hadoop that rely on the capability of multiple JVM executions: first, TaskTracker and DataNode are running two different JVM processes; second, for each task processed in a TaskTracker node, a new JVM instance is launched unless there is an idle JVM which can be reused for the task.

To support these behaviors while coping with the STB limitation of running only one JVM instance, we implemented a new `ProcessBuilder` class that creates a thread group whenever the launch of a new JVM process is requested. Each thread group provides a distinct set of Hadoop environmental variables which are managed within the threads belonging to a given thread group without interfering with other threads groups. The `ProcessBuilder` class implementation also enables optimizations such as replacing IPC (Inter-Process Call) with method invocations in the same process, and the elimination of local data transfers through sockets with local file-copy operations.

A number of other middleware issues related to porting Hadoop to an embedded device like the STB were discovered and resolved. For example, certain Java classes have bugs that make the application behave improperly, halt, or sometimes fail. In these cases the classes were replaced with better implementations or patched to align with the Hadoop Java class requirements. Also, some configuration changes were made to the system: e.g., the Socket timeout constant had to be slightly extended to account for variations in network response times or delays. Finally, to relieve memory constraints, we reduced the number of threads associated to unimportant services such as the metrics service which profiles the statistics of performance or the web service that provides status information.

IV. Experiments

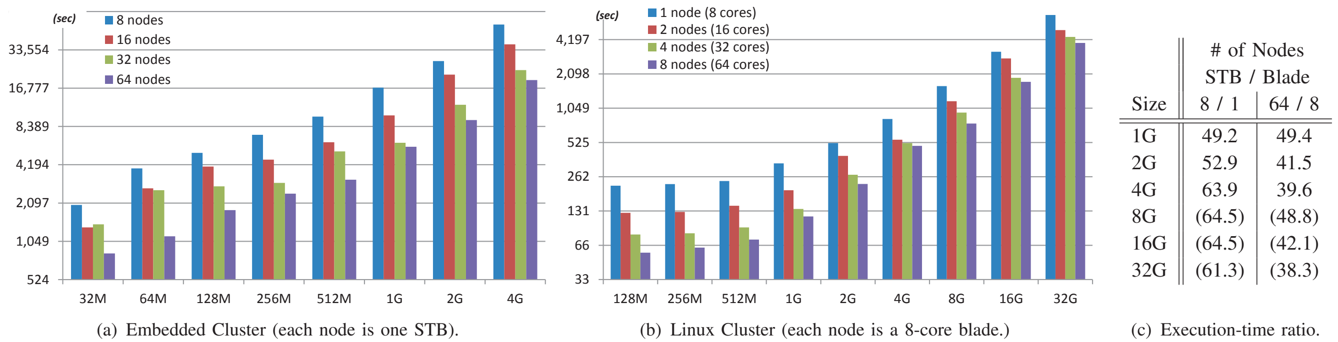
In order to evaluate our embedded Hadoop system for its scalability characteristics and execution performance, we executed a number of MapReduce experimental tests across the Linux Cluster and Embedded Cluster. All the experiments were performed while varying the degree of parallelism, i.e. by iteratively doubling the number of Hadoop nodes, of each cluster: specifically, from 1 to 8 Linux blades for the Linux Cluster (where each blade contains eight 2GHz processor cores) and from 8 through 64 STBs for the Embedded Cluster (where each STB contains one 400MHz processor core). The results can be organized in four groups which are presented in the following subsection. We report the average results after executing all tests multiple times.

A. The WordCount Application

WordCount is a typical MapReduce application that counts the occurrences of each word in a large collection of documents. The results reported in Fig. 5(a) and 5(b) show that this application scales consistently for both the Embedded Cluster and Linux Cluster. As the size of the input data increases, the Embedded Cluster clearly benefits from the availability of a larger number of STB nodes to process larger data sets. The Linux Cluster execution time remains approximately constant for data sizes growing from 128MB to 512MB since these are relatively small, but then it begins to double as the data sizes grow from 1GB to 32GB. In fact, above the 1GB threshold the amount of data that needs to be shuffled in the Reduce task begins to exceed the space

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society



Size	# of Nodes STB / Blade 8 / 1	64 / 8
1G	49.2	49.4
2G	52.9	41.5
4G	63.9	39.6
8G	(64.5)	(48.8)
16G	(64.5)	(42.1)
32G	(61.3)	(38.3)

(a) Embedded Cluster (each node is one STB). (b) Linux Cluster (each node is a 8-core blade.) (c) Execution-time ratio.

Figure 5. *WordCount* execution time as function of problem size (bytes), node count: (a) Embedded Cluster and (b) Linux Cluster; (c) relative comparison.

available within the heap memory of each node. A similar transition from in-memory shuffling to in-disk shuffling occurs in the Embedded Cluster for smaller data sets due to the smaller memory available in the STB nodes: specifically, it occurs somewhere between 64MB and 512MB, depending on the particular number of nodes of each Embedded Cluster configuration.

Fig. 5(c) reports the ratios between the execution times of two Embedded Cluster configurations over two corresponding equivalent Linux Cluster configurations, for large input data sets. [2] The first column reports the ratio of the configuration with eight STBs over one single blade with eight processor cores; the second column reports the ratio of the Embedded Cluster configuration (with 64 STBs) over the Linux Cluster configuration (with eight blades for a total of 64 cores.) Across the different data sizes, the performance gap of the Embedded Cluster relative to the corresponding Linux Cluster with the same number of Hadoop nodes remain approximately constant: it is about 60 times slower for the configuration with 8 nodes and about 40 times slower for the one with 64 nodes. Notice that *these values are the actual measured execution times; they are not modified to account for the important differences among the two systems such as the 5X gap in the processor's clock frequency between the Linux blades and the STBs.* A comprehensive discussion of the reasons behind the performance gap between the two systems and how this may be reduced in the future is given in Section IV-E.

B. HDFS & MapReduce Benchmarks

The second group of experiments involve the execution of a suite of standard Hadoop benchmarks. The goal is to compare how the performance of the Embedded Cluster and Linux Cluster scales for different MapReduce applications. The execution times of these applications expressed in seconds and measured for different configurations of the two clusters are reported in Table 2. The numbers next to the application names in the first column denote input parameters, which are specific to each application:

[2]The values in parenthesis are computed by extrapolating the execution times on the Embedded Cluster.

Benchmarks	8 STBs (8 cores)	64 STBs (64 cores)	1 Blades (8 cores)	8 Blades (64 cores)
Sleep	1285.1	119.6	1223.6	114.5
RandomTextWriter 8	799.6	743.9	177.6	172.0
PiEstimator 1k	461.1	163.5	212.1	52.5
PiEstimator 16k	463.4	474.0	213.7	52.5
PiEstimator 256k	603.6	783.2	214.6	52.4
PiEstimator 4M	1240.9	2048.2	213.9	52.5
PiEstimator 64M	7373.0	10482.5	314.8	58.4
K-Means 1G	3679.2	1149.3	794.7	24.5
Classification 1G	3009.0	784.9	864.7	25.45

Table 2. Execution times (in seconds) for various Hadoop benchmarks.

e.g. "RandomTextWriter 8" denotes that the RandomTextWriter application is running eight mappers, while the "PiEstimator 1k" means that Pi-estimator runs with a 1k sample size.

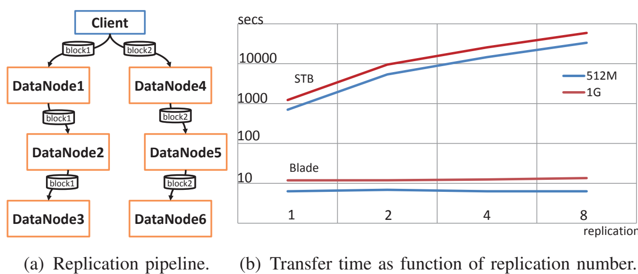
Sleep is a program that simply keeps the processor in an idle state for one second, whenever a Map or a Reduce task should be executed. Hence, this allows us to estimate the performance overhead of running the Hadoop framework. For the representative case of running Sleep with 128 mappers and 16 reducers, the Embedded Cluster and the Linux Cluster performance is basically the same.

RandomTextWriter is an application that writes random text data to HDFS and, for instance, it can be configured to generate a total of 8GB of data uniformly distributed across all the Hadoop nodes. When it is running, eight mappers are launched on each Linux blade, i.e. one per processor core, while only one mapper is launched on each STB node. Since the I/O write operations dominate the execution time of this application, scaling up the number of processor cores while maintaining the size of the random text data constant does not really improve the overall execution time.

Pi-Estimator is a MapReduce program that estimates the value of the π constant using the Monte-Carlo method [6]. For the Linux Cluster, the growth of the input size does not really impact the execution time for a given system configuration, while moving from a configuration with one blade to one with eight blades yields a 4x speedup. For the Embedded Cluster, in most cases scaling up the number of nodes causes higher execution times because this program requires that during the initialization phase the STBs receive a set of large class files which are not originally present in

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society



(a) Replication pipeline. (b) Transfer time as function of replication number.

Figure 6. HDFS data-replication mechanism (R=3) and replication time.

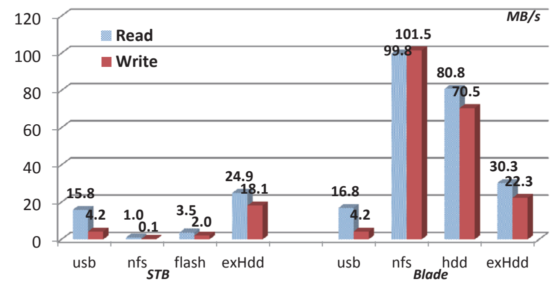


Figure 7. Native IO Performance Comparison

the Embedded Java Stack. This file transfer, which uses the pipelined mechanism explained in Section IV-D, takes a long time that more than cancel out any benefits of increasing the number of Hadoop nodes.

C. Data Mining Applications

To evaluate the feasibility of utilizing the Embedded Cluster system for data mining applications, we performed two experiments based on MapReduce versions of two common algorithms. *K-Means* is a popular data mining algorithm to cluster input data into K clusters: it iterates until the change in the centroids is below a threshold to successively improve the clustering result [14]. *Classification* is a MapReduce version of a classic machine learning algorithm: it classifies the input data into one of K pre-determined clusters [20]. Unlike K-Means, Classification does not run iteratively, and, therefore, does not produce intermediate data.

The last two rows in Table 2 report the results of running these two applications, each with an input data set of size 1GB. For both applications the results are similar: the execution time when running on the Embedded Cluster with eight STBs is about four times longer than running in the Linux Cluster with one 8-core blade; furthermore, when both systems are scaled up by a factor of eight, the performance gap grows from four to forty times. The growing gap is mainly due to the fact that scaling up the system parallelism while keeping the input data size constant leads to shuffling a large number of small data sets across the Hadoop nodes. This requires peer-to-peer communication among the nodes, an operation that the DOCSIS network of the Embedded Cluster does not support as well as the gigabit Ethernet network of the Linux Cluster does. To better evaluate the difference in transfer time between the two networks we complete the following experiment focused on the HDFS data replication, which requires similar peer-to-peer communication among the Hadoop nodes.

D. Data Replication in HDFS

The Hadoop Distributed File System (HDFS) replicates data blocks through pipelining of DataNodes based on the scheme illustrated in Fig. 6(a): for a given replication number R, a pipeline of R DataNodes is created whenever a new block is copied to a DataNode and the data are transferred to the next DataNode in the pipeline until the last one receives it. This mechanism causes a large transfer-time penalty for the Embedded Cluster due to DOCSIS-network overhead associated with the transfer of data between pairs of Hadoop nodes. Specifically, a DOCSIS network does not support direct point-to-point communications among STBs. Instead, all communications occur between a given STB and the DOCSIS router located in the cable-system head-end: this acts as a forwarding agent on behalf of the two communicating STBs. Due to this architecture, as we increase the number of STBs in the system (each STB corresponding to one Hadoop node) more slow communications between pairs of STBs occur, thus impacting negatively the overall data-replication time. In contrast, the data replication time spent in the Linux Cluster remains constant as we grow the number of nodes thanks to: (i) the fast communication channels among cores on the same blade and (ii) the gigabit Ethernet network connecting cores across different blades.

E. Discussion

The performance of executing Hadoop MapReduce applications is influenced by various system properties including: the processor speed, memory, I/O, and networking capabilities of each node. Further, the relative impact of each factor depends on the computation and communication properties of the specific MapReduce application in a way that may vary considerably with the given input problem size and the total number of nodes comprising the Hadoop system. Next, we discuss how the system properties of the Embedded Cluster compare to those of the Linux Cluster and outline how the technology trends may reduce the gap between the two systems.

Processor performance. In our experimental setup, there is a 5X gap in processor clock frequency between the Embedded Cluster and Linux Cluster nodes. Further, we empirically noticed another factor of 2X in processing speed which we attributed to the different computer architectures of the 2GHz Xeon and 400MHz MIPS processors. This gap is expected to decrease considerably as next-generation STB devices will incorporate commodity 1GHz+ multi-core processors now found in smartphone and tablets, while it is unlikely that the blade clock frequency will increase much.

I/O Operations. The *RandomTextWriter* benchmark represents many MapReduce applications which execute numerous data-block storage operations. In fact, the Hadoop

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

system itself can be very I/O intensive when performing data replication. We run the *TestDFSIO* test to evaluate the I/O performance of HDFS by reading/writing files in parallel through a MapReduce job. This program reads/writes each file in a separate map task, and the output of the map is used for collecting statistics relating to the file just processed; then, the statistics are aggregated in the reduce task to produce a summary. The results of running TestDFSIO reveal that an STB has 0.115MB/s reading and 1.061MB/s writing speed while the corresponding values for a Linux blade are 68.526MB/s and 99.581MB/s. [3] We also run a simple native C program that executes read/write operations using large files on the two clusters with 4 different interfaces: USB, FLASH, NFS, and HDD. The results are reported in Fig. 7. We note that the network performance of STB NFS reads/writes is significantly less, by a factor of nearly 100, than the network performance of the Linux blade server. This gap is primarily due to the DOCSIS network, whose effective transfer rate is limited to 4MB/s compared to 1Gb/s Ethernet network, whose effective maximum transfer rate is closer to 125MB/s. On the other hand, the measured performance of the USB and external hard-drive interfaces on both the STB and Linux blade server is comparable. This is due to the common commodity SoC for USB and disk interfaces used in the design of both the STBs and blades. In our experiments, the Linux blades use an internal hard-drive disk (HDD) while the STBs, which do not contain an internal hard-drive, rely on a USB memory stick whose read performance is 6 times slower (and write performance is 24 times slower) than the HDD when providing HDFS storage. This gap can be reduced by having the STBs use a better file system for the USB sticks than FAT32 such as SFS [19]. Also, as shown in Fig. 7, an external USB HDD could provide a 1.5-4.2 speed-up for reading/writing over the USB memory stick. Here, the technology trends should provide next-generation STB devices with HDD and USB 3.0.

Networking. The lack of support for peer-to-peer communication among STBs in the DOCSIS network limits considerably the HDFS replication mechanism (as discussed in Section IV-D), the Hadoop shuffling operations (as seen for the K-Means, Classification and WordCount programs), and the transfer of large class files during the initialization phase (as in the PI-Estimator). In particular, shuffling generates an implicit all-to-all communication pattern among nodes that is application specific: each node sends its corresponding Map results to other nodes through HTTP, generating $|Node|^2$ communication exchanges, which for the DOCSIS network results in inefficient upstream communication requests as nodes attempt to transfer data blocks from Mappers to Reducers. A similar performance impact occurs during Hadoop replication: for a given replication factor R and a total number of blocks M, the number of DOCSIS upstream communication transfers to complete replication is $M \times (R-1)$. As the input size increases the number of blocks increases in direct proportion, thus increasing the replication time. The scalability in Embedded Cluster largely depends on the amount of data to be shuffled generated by the Map tasks and the replication communication overhead. This problem may be addressed in part with the deployment of the higher performance DOCSIS 3.0 standard [12], which supports up to 300 Mb/s upstream bandwidth. Then, opportunities for further improvements include: optimization of the Hadoop scheduling policy, network topology optimization, and leveraging the inherent multi-casting capabilities of DOCSIS to reorder the movement of data blocks among nodes and reduce network contention.

[3]The STB shows significant difference between upload and download speed due to the inherently asymmetric and lower transfer rate characteristics of the DOCSIS network.

V. Related Work

The Hadoop platform for executing MapReduce applications has received great interest in recent years as problems in large-scale data analysis and Big Data have increased in importance. Work in the area of heterogeneous MapReduce computation, however, remains rather limited, notwithstanding the growth of embedded devices interconnected through broadband networking to distributed data centers. Our work is aligned with efforts in the Mobile Space to bridge MapReduce execution to embedded systems and devices. For example, the Misco system implements a novel framework for integrating smartphone devices for MapReduce computation [10]. Similarly, Elespuro *et al.* developed a system for executing MapReduce using smartphones under the coordination of a Web-based service framework [11]. Besides the fact that our system uses a wired network of embedded stationary devices instead of a mobile network, the main difference with these systems is that we ported the Hadoop framework, including the HDFS, based on the Java programming model. Other related work include utilizing GPU processors to execute MapReduce [15]. While most related work in adapting MapReduce execution to embedded devices has focused on leveraging service-side infrastructure, our work is closer to current research under way for large scale execution of MapReduce applications on the Hadoop platform across Linux blades and PC clusters [25].

VI. Conclusion

We developed, implemented, and tested a heterogeneous system to execute MapReduce applications by leveraging a broadband network of embedded STB devices. In doing so, we addressed various general challenges to successfully port the Hadoop framework to the embedded JVM environment. We completed a comprehensive set of experiments to evaluate our work by comparing various configurations of the prototype Embedded Cluster with a more traditional Linux Cluster. First, the results validate the feasibility of our

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

idea as the Embedded Cluster successfully executes a variety of Hadoop applications. From a performance viewpoint, the Embedded Cluster typically trails the Linux Cluster, which can leverage more powerful resources in terms of processor, memory, I/O, and networking. On the other hand, for many applications both clusters demonstrate good performance scalability as we grow the number of Hadoop nodes. But a number of problems remain to be solved to raise the performance of executing MapReduce applications in the Embedded Cluster: in particular, critical areas of improvement include the STB I/O performance and the communication overhead among pairs of STBs in the DOCSIS broadband network. Still, the gap between embedded processors and blade processors in terms of speed, memory, and storage continues to decrease, while higher performance broadband networks are expected to integrate embedded devices into the Cloud. These technology trends hold the promise that future versions of the MapReduce computing system presented in this paper can help to leverage embedded devices for Internet-scale data mining and analysis.

ACKNOWLEDGMENT

This project is partially supported by Cablevision Systems.

REFERENCES

[1] "http://backport-jsr166.sourceforge.net."
[2] "http://hadoop.apache.org."
[3] "http://retrotranslator.sourceforge.net."
[4] "http://www.cablelabs.com."
[5] "http://www.tru2way.com."
[6] J. Arndt and C. Haenel, *Pi-Unleashed*. Springer, 2001.
[7] S. Bangay, "Experiences in porting a virtual reality system to Java," in *Proc. of the 1st Intl. Conf. on Comp. Graphics, Virtual Reality and Visualisation*, pp. 33–37, Nov. 2001.
[8] I. D. Corporation, "Extracting value from chaos," Jun. 2011.
[9] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[10] A. Dou *et al.*, "Misco: a MapReduce framework for mobile systems," in *Proc. of the 3rd Intl. Conf. on Pervasive Tech. Related to Assistive Environments*, pp. 32:1–32:8, Jun. 2010.
[11] P. R. Elespuru, S. Shakya, and S. Mishra, "MapReduce system over heterogeneous mobile devices," in *Proc. of the 7th IFIP WG 10.2 Intl. Workshop on SW Tech. for Embedded and Ubiquitous Sys.*, pp. 168–179, Nov. 2009.
[12] F. Gatta *et al.*, "An embedded 65 nm CMOS baseband IQ 48 MHz-1 GHz dual tuner for DOCSIS 3.0," *Comm. Mag.*, vol. 48, no. 4, pp. 88–97, Apr. 2010.
[13] T.-M. Grønli, J. Hansen, and G. Ghinea, "Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments," in *Proc. of the 3rd Intl. Conf. on Pervasive Tech. Related to Assistive Env.*, pp. 45:1–45:8, Jun. 2010.
[14] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, Mar. 1979.
[15] B. He *et al.*, "Mars: a MapReduce framework on graphics processors," in *Proc. of the 17th Intl. Conf. on Parallel Arch. and Compilation Tech.*, pp. 260–269, Oct. 2008.
[16] S. Kågström, H. Grahn, and L. Lundberg, "Cibyl: an environment for language diversity on mobile devices," in *Proc. of the 3rd Intl. Conf. on Virtual execution environments*, pp. 75–82, Jun. 2007.
[17] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 938–948, Jan. 2010.
[18] F. P. Miller, A. F. Vandome, and J. McBrewster, *BD-J*. Alpha Press, 2009.
[19] C. Min *et al.*, "SFS: Random write considered harmful in solid state drives," in *Proc. of the 10th USENIX Conf. on File and Storage Tech.*, Feb. 2012.
[20] C. Moretti, K. Steinhaeuser, D. Thain, and N. Chawla, "Scaling up classifiers to cloud computers," in *8th IEEE Intl. Conf. on Data Mining*, pp. 472–481, Dec. 2008.
[21] R. Neill *et al.*, "Embedded processor virtualization for broadband grid computing," in *Proc. of the IEEE/ACM 12th Intl. Conf. on Grid Computing*, pp. 145–156, Sep. 2011.
[22] R. Neill, A. Shabarshin, and L. P. Carloni, "A heterogeneous parallel system running open MPI on a broadband network of embedded set-top devices," in *Proc. of the ACM Intl. Conf. on Computing Frontiers*, pp. 187–196, May 2010.
[23] R. Pinilla and M. Gil, "ULT: a Java threads model for platform independent execution," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 4, pp. 48–62, Oct. 2003.
[24] M. Schoeberl, S. Korsholm, T. Kalibera, and A. P. Ravn, "A hardware abstraction layer in Java," *ACM Trans. Embedded Comp. Sys.*, vol. 10, no. 4, pp. 42:1–42:40, Nov. 2011.
[25] N. Thanh-Cuong, S. Wen-Feng, C. Ya-Hui, and X. Wei-Min, "Research and implementation of scalable parallel computing based on map-reduce," *Journal of Shanghai Univ.*, vol. 15, no. 5, pp. 426–429, Aug. 2011.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE