Data Analysis for Massively Distributed Simulations

Ke-Thia Yao, Robert F. Lucas, Craig E. Ward, Gene Wagenbreth
USC Information Sciences Institute
Marina del Rey, California
{kyao,rflucas,cward,genew}@isi.edu

Thomas D. Gottschalk
California Institute of Technology
Pasadena, California
tdg@cacr.caltech.edu

ABSTRACT

More computing power allows increases in the fidelity of simulations. Fast networking allows large clusters of high performance computing resources, often distributed across wide geographic areas, to be brought to bear on the simulations. This increase in fidelity has correspondingly increased the volumes of data simulations are capable of generating. Coordinating distant computing resources and making sense of this mass of data is a problem that must be addressed. Unless data are analyzed and converted into information, simulations will provide no useful knowledge. This paper reports on experiments using distributed analysis, particularly the Apache Hadoop framework, to address the analysis issues and suggests directions for enhancing the analysis capabilities to keep pace with the data generating capabilities found in modern simulation environments. Hadoop provides a scalable, but conceptually simple, distributed computation paradigm based on map/reduce operations implemented over a highly parallel, distributed filesystem. We developed map/reduce implementations of K-Means and Expectation-Maximization data mining algorithms that take advantage of the Hadoop framework. The Hadoop filesystem dramatically improves the disk scan time needed by these iterative data mining algorithms. We ran these algorithms across multiple Linux clusters over specially reserved high speed networks. The results of these experiments point to potential enhancements for Hadoop and other analysis tools.

ABOUT THE AUTHORS

Ke-Thia Yao is a project leader and research scientist in the University of Southern California Information Sciences Institute. He has worked developing a suite of monitoring/logging/analysis tools to help users better understand the computational and behavioral properties of large-scale simulations. He received his B.S. degree in EECS from UC Berkeley, and his M.S. and Ph.D. degrees in Computer Science from Rutgers University.

Robert Lucas is the Director of the Computational Sciences Division at the Information Sciences Institute (ISI). There he manages research in computer architecture, VLSI, compilers and other software tools. He has been the principal investigator on the JESPP project since its inception in 2002. Dr. Lucas received his BS, MS, and PhD degrees in Electrical Engineering from Stanford University in 1980, 1983, and 1988 respectively.

Craig E. Ward is a Parallel Computer Systems Analyst at the Information Sciences Institute. Much of his recent research has focused on large-scale data management. He has a B.A. in History from the University of California, Irvine, and an M.S. in Computer Science from Loyola Marymount University.

Gene Wagenbreth is a Systems Analyst for Parallel Processing at the Information Sciences Institute, doing research in the Computational Sciences Division. He has also been active in benchmarking, optimization and porting of software for private industry and government labs. He received a BS in Math/Computer Science from the University of Illinois in 1971.

Thomas D. Gottschalk is a Member of the Professional Staff, Center for Advanced Computing Research (CACR) and Lecturer in Physics at the California Institute of Technology. He received a B.S. in Physics from Michigan State University and a Ph.D. in Theoretical Physics from the University of Wisconsin.

Ke-Thia Yao, Robert F. Lucas, Craig E. Ward, Gene Wagenbreth
USC Information Sciences Institute
Marina del Rey, California
{kyao,rflucas,cward,genew}@isi.edu

Thomas D. Gottschalk
California Institute of Technology
Pasadena, California
tdg@cacr.caltech.edu

INTRODUCTION

More computing power allows increases in the fidelity of simulations. Fast networking allows large clusters of high performance computing resources, often distributed across wide geographic areas, to be brought to bear on the simulations. This increase in fidelity has correspondingly increased the volumes of data that simulations are capable of generating.

Coordinating distant computing resources and making sense of this mass of data is a problem that must be addressed. Unless data are analyzed and converted into information, simulations will provide no useful knowledge. For the US Joint Forces Command (USJFCOM) Urban Resolve exercises we developed a distributed logging system to capture publish/subscribe messages from the High-Level Architecture (HLA) simulation federation. For a two-week exercise, omitting nonessential data, we logged over a terabyte of data [Yao & Wagenbreth 2005].

This paper reports on experiments using distributed analysis, particularly the Apache Hadoop framework, to address the analysis issues and suggests directions for enhancing the analysis capabilities to keep pace with the data generating capabilities found in modern simulation environments. Hadoop provides a scalable, but conceptually simple, distributed computation paradigm based on map/reduce operations implemented over a highly parallel, distributed filesystem. We developed map/reduce implementations of K-Means and Expectation-Maximization data mining algorithms that take advantage of the Hadoop framework. The Hadoop filesystem dramatically improves the disk scan time needed by these iterative data mining algorithms. We ran these algorithms across multiple Linux clusters over specially reserved high speed networks. The results of these experiments point to potential enhancements for Hadoop and other analysis tools.

Data mining Hadoop jobs were created to experiment with the performance characteristics of Hadoop in an environment that provided high-speed network connections to sites across large geographic regions. High performance Linux Cluster computers were installed at the Information Sciences Institute (ISI) in California, at the University of Illinois – Chicago (UIC) in Illinois, and ISI East in Virginia. The machine at ISI served as a control. Special network connectivity was established between UIC and ISI East to test Hadoop across a great geographic distance.

OVERVIEW OF HADOOP

Hadoop is an open source system, hosted by the Apache Software Foundation that provides a reliable, fault tolerant, distributed file system and application programming interfaces. These enable its map-reduce framework for analyzing large volumes of data in parallel.

We found that the simplicity of the Hadoop programming model allows for straightforward implementations of many applications. Java applications have the most direct access, but Hadoop also has streaming capabilities that allow for implementations in any preferred language.

Several organizations that need to handle large amounts of data are using map-reduce implementations to manage that data. Google started using a map-reduce system internally before 2004 [Dean 2004]. Yahoo runs the largest Hadoop cluster, running over a Linux cluster of over 10,000 cores [Yahoo 2008]. Vendors, such as Amazon, utilize Hadoop as part of their cloud computing service. A growing list of organizations making use of Hadoop can be found at the Hadoop wiki [Powered By, 2009]. In the 2008 terabyte sort challenge, Yahoo won by using Hadoop to sort 1 terabyte of data in 209 seconds [O'Malley 2008]. That cluster consisted of 910 nodes with 2 quad core 2GHz Xeons and 4 SATA disks per node.

Hadoop Distributed File System

The Hadoop Distributed Files System (HDFS) runs on top of a native file system and is only accessible through the Hadoop Application Programming Interfaces (APIs).

HDFS configurations distribute data in equally sized chunks across the available data nodes. This division of data works best for large files that can be stored as

multiples of the chunking size configured for the HDFS. If the files are smaller than the chunking size, the HDFS will waste local file system resources with empty, allocated bytes.

Redundancy and fault tolerance are achieved by replicating these chunks on multiple nodes. Hadoop attempts to run the map operations on copies of the data local the mapping task. This reduces the amount of data that needs to be moved around.

Our experiments used varying HDFS configurations. One configuration kept all nodes within a single rack. Another divided the nodes across half of the continental United States.

Map-Reduce API

Hadoop exposes three operations for implementing the map-reduce algorithm, mapping, combining, and reducing. The system is implemented in Java; however, Hadoop also exposes a streaming interface that allows programs written in any language to process each operation.

The data are divided into chunks by the HDFS. Each map operation executes on a chunk of data, usually stored nearby. As the mapper iterates over the chunk, it assigns values to key elements. These key/value pairs may then be passed to a combine operation to collect the keys. A reduce operation combines the values for each key.

A simple example is counting words in files of English text.

As each file is processed, each word becomes a key with the value the count of how many times the word appeared in the file. These key value pairs, the words and associated counts, are sorted and passed to combine operation. (In this simple example, the combine step does not do anything significant. The K-Means jobs used for the experiments did take advantage of the operation.) Finally, the combined pairs are reduced with each key assigned the sum of the values of the preceding operations. The tutorial included with the Hadoop documentation goes into more detail.

Hadoop Node Types

Hadoop has three different node types: nodes for processing tasks, nodes for storing data, and a single node, called the name node, to coordinate the others. The tasks that are assigned to processing nodes are monitored for status. If a task appears to fail, it can be reassigned to another processing node. The assignments attempt to keep processing and data near each other, limiting the strain on any underlying communications resources, such as a network.

DISTRIBUTED DATA MINING ALGORITHMS

Data mining is a way of finding patterns in what otherwise would be random data. Many data mining algorithms are iterative in nature. They require the data to be scanned several times during the mining process. These algorithms can become prohibitively expensive for very large data sets that do not fit into memory, and have to be stored on disk. Sequential disk access on a single disk can be several orders of magnitude slower than memory access. Hadoop with its potential to access thousands of disks in parallel provides a way of addressing this problem.

In addition, in some situations the data themselves are stored in a distributed fashion. For example, for JFCOM's Urban Resolve exercises, we implemented a distributed logger that stored High-Level Architecture Runtime Infrastructure (HLA RTI) messages locally where the messages were emitted [Yao & Wagenbreth 2005; Graebener *et al* 2003]. Using Hadoop provides a convenient way to process the data without having to move it to a centralized location.

Two Clustering Algorithms

To test the feasibility of this approach we implement two data mining clustering algorithms in Hadoop: K-Means and Expectation-Maximization (EM).

K-Means is a popular data mining clustering algorithm that assigns a set of data instances into clusters (or subsets) based on some similarity metric. The K-Means algorithm requires three inputs: an integer k to indicate the number of desired clusters as output, a distance function over the data instances, and the set of n data instances to be clustered. The distance of a data instance to itself is zero. The greater the distance between two data instances, the less similar the instances are. Typically, a data instance is represented as a numerical *vector*. The output of the algorithm is a set of k points representing the mean (or the center) of the k clusters. Each of the n data instances is assigned to the nearest cluster mean based on the distance function.

Here is pseudo code for the K-Means algorithm:

1. Generate an initial guess for the *k* cluster (for example, by randomly selecting *k* points from the data instances as the *k* means).
2. Assign each of the *n* data instances to the nearest cluster mean.
3. Based on the data instance assignment, compute the new cluster mean for each of the *k* clusters.
4. While not done, go to Step 2.

Figures 1 illustrates some results of K-Means clustering. Figure 1 shows K-Means correctly finding the means of the 3 distinct clusters. That is, given a set of points generated for this dataset, the algorithm correctly discovered the patterns in the points.

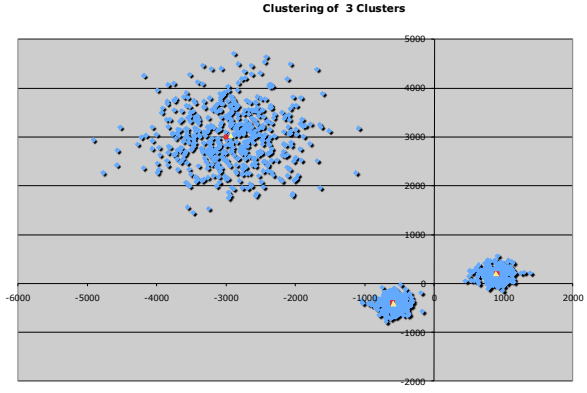

Figure 1 K-Means clustering of three distinct clusters of points.

The EM algorithm can be viewed as a probabilistic generalization of the K-Means algorithm. Instead of representing a cluster by just its mean, EM represents a cluster by its mean and its variance (or covariance matrix), *i.e.* each cluster is represented by a Gaussian distribution. In addition, each cluster is associated with a weight, representing the probability of selecting the cluster. The sum of these *k* cluster weights is equal to one. This representation is called a Gaussian mixture model.

The steps of the EM algorithm are similar to the K-Means algorithm. In Step 1 the initial guess now includes the *k* means, *k* variances, and *k* cluster weights. The assignment in Step 2, also known as the Expectation Step, is now slightly more complicated. Instead of assigning each data point to one cluster, each data point is assigned to each cluster with a probability based on a Gaussian distribution. In Step 3, the Maximization Step, the *k* means, *k* variances, and *k* cluster weights are recomputed based on the probabilistic assignment from Step 2.

Hadoop Implementation

We shall only describe the Hadoop implementation of the K-Means algorithm. The structure of the EM Hadoop implementation is similar.

There exists a variety of ways to generate the initial guess in Step 1. If there is *a priori* knowledge of the range of possible values of the data instance attributes, then we can generate *k* means randomly using a uniform distribution. Otherwise, we can scan the data instances once to compute the range values. Or, we can scan the data instances and randomly select *k* instances as the means. To simplify the algorithm description we shall assume there is *a priori* knowledge.

Step 1:
generate initial guess

Step 2:
corresponds to the map operation. Map functions have the form:

Map: *(in-key, in-value)* → *list (out-key, out-value).*

In this case, the *in-key* is null, and the *in-value* is the data instance vector. The *out-key* is an integer from 1 to *k* representing the cluster identifier, and the *out-value* is a list of pairs, where each pair consists of the data instance vector and the integer one.

K-Means Map: *(null, data-instance)* → *list (cluster-id, (data-instance, 1))*

Step 3:
corresponds to the reduce operation. Reduce functions have the form:

Reduce: *(in-key, list (in-value))* → *list (out-key, out-value>*

In this case the input *(in-key, in-value)* is the output of the K-Means Map (*cluster-id, (data-instance, 1)).* For each *cluster-id*, the reduce operation sums all the *(data-instance, 1)* pairs associated with that cluster-id.

K-Means Reduce: *(cluster-id, (data-instance, count))* → *list (cluster-id, (sum-of- data-instances, number-of-instances))*

Here the *sum-of- data-instances* divided by *number-of-instances* is the mean of the cluster.

Here is a simple, but naïve, Hadoop implementation of the K-Means algorithm:

1. Random generate *k* points as initial *k* means.
2. Apply K-Means Map & Reduce.
3. While not done, go to Step 2.

A slightly more sophisticated Hadoop implementation would add a Combine operation in between the Map and the Reduce. In Hadoop the Map and the Reduce operations typically reside on different compute nodes. This naïve implementation would pass all *n* data instance pairs across the network from Map to Reduce. The Combine operation would reduce the amount of data that has to be transferred across the network.

K-Means Combine: *(cluster-id, (data-instance, count)) → list (cluster-id, (partial-sum-of- data-instances, partial-count-number-of-instances))*

EXERCISING HADOOP

Test Environment Setup

Data mining Hadoop jobs were created for the SIMC-IC project to experiment with the performance characteristics of Hadoop in an environment that provided high-speed network connections to sites across large geographic regions. As mentioned before, clusters in California, Illinois and Virginia were connected via a high-bandwidth link.

Each cluster machine was comprised of:

- 10 nodes
- 5.3 TB local disk
- 2 Clusters running Fedora 8
- 1 Cluster running Debian
- 1 10GigE network card
- 1 1Gig card for management only
- Dual Quad Core (8 cores per node) CPUs

The version of Hadoop used for the experiments was 0.17.2.1. Each cluster used the Java SE Runtime Environment 1.6 (build 1.6.0_11-b03).

Hadoop clusters were configured using the available nodes such that both the control Hadoop cluster and the distributed Hadoop cluster had the same number of nodes, one name node and nine nodes running data and job task services. The only difference being that the control cluster used only local network connections while the other used wide area network connections.

For the wide area network Hadoop cluster, two configurations were used. One configuration used the default network resources and one used dedicated Internet 2 high-bandwidth lines reserved for short time periods.

Data Load

In addition to the data mining jobs developed, the ability of Hadoop to load and store data was tested. A simple data load of six 1.2-gigabyte files was performed using the default settings, each block of data replicated on three nodes.

Data Load Test Results

All time data was collected from the *time(1)* command.

Table 1: Data Load Test Results

	User	System	Elapsed
ISI Local	44.85	22.09	2:05.69
ISIE/UIC (standard)	46.98	18.38	14:27.75
ISIE/UIC (fastnet)	49.18	18.94	29:20.78

As would be expected, the quickest data loads were with the local nodes configuration. The actual processing times were not that much different for each configuration. The major difference was in clock time indicating that the distributed systems spent significant time in suspended wait states while the network subsystems performed their functions. The Fastnet version using Internet 2 actual took longer elapsed time than the standard version. However, during the execution of the Fastnet version, we did observe Java network exceptions being thrown. We will address this anomaly in the next section.

Data Mining Jobs

Two implementations of the K-Means algorithm were used to test the processing capabilities of Hadoop. An expectation-maximization job was also developed, but this job was not used for this experiment. The UIC Angle dataset was searched for points within the data where the data clustered. One implementation used a "naïve" approach while the other used a more efficient, "smart" approach. The naïve implementation did not use the combine step allowed by the Hadoop API. This resulted in much more network usage as more data had to be passed around between the task nodes. The smart implementation made use of this step and greatly reduced the amount of data exchanged.

The K-Means jobs iterated over the data set with an initial set of cluster points, each time updating the set of cluster points to better fit the data, each resulting set of cluster points becoming the input for the next iteration. When either the points stopped significantly changing or the maximum number of iterations was reached, the job stopped.

For development and initial testing, the job was tested using points randomly generated using known center coordinates. The results of a run were expected to match the input provided to the random point generator.

Table 2: K-Means Results

	User	System	Elapsed
ISI Local (smart)	1.68	0.18	1:37.76
ISI Local (naïve)	6.55	0.92	40:38.64
ISIE/UIC (smart/stand)	1.67	0.19	1:52.80
ISIE/UIC (smart/fastnet)	2.25	0.27	8:25.08
ISIE/UIC (naive/stand)	5.35	0.96	1:12:03
ISIE/UIC (naive/fastnet)	8.40	1.72	2:14:16 KILLED[1]

As with the data loads, the data mining jobs performed best on the local nodes setup. The differences between local and networked systems are not as pronounced as with the data loads. This is likely due to the ability of Hadoop to process chunks of data in a "rack-aware" manner. The smart implementations tended to not require long haul network services and were able to process data in what to them was a local manner. Again, the Fastnet version took longer elapsed time than the standard version. We will address this anomaly in the next section.

NETWORK UTILIZATION

In the previous section our experiments exercised Hadoop across differing network configurations. One configuration used the "normal" connectivity found in the network while another ran Hadoop over special high-speed links with a theoretical peak throughput of 10 Gbps. But, Hadoop results did not reflect the advantage of the high-speed links.

To rule out the possibility the high-speed links were faulty we used another software system to get independent measurements. The tool used to test this capability was the Meshrouter, which was designed for high throughput HLA RTI communications [Barrett & Gottschalk 2004; Brunett & Gottschalk 1998]. The tests show the Meshrouter application is capable of achieving 1.5Gbps with a single TCP stream, and up to 5 Gbps with combined streams.

Based on this throughput experiment we reasoned that Hadoop is not able to take advantage of the high-speed network. As mentioned previously we observed Java network exceptions during the execution. Although Hadoop is designed to be fault tolerant, the exceptions most likely slow downed its execution.

Moreover, in order to achieve 50% capacity of the high-speed network, the Meshrouter application required several TCP streams. We suspect that even without the network exceptions Hadoop will not be able to take full advantage of the high-speed network.

Below, we describe the details of the high-speed network throughput experiment using the Meshrouter. The Meshrouter and associated applications implement interest managed communication (RTI) utilized by several entity simulators in general use. Test programs named publish and subscribe were used to exercise the network in a controlled and repeatable manner. The Meshrouter is a complex real-world application.

The bandwidth experiments were done using the standard ISI MeshRouter formalism for interest-managed communications. A schematic of the MeshRouter is shown in Figure 3.

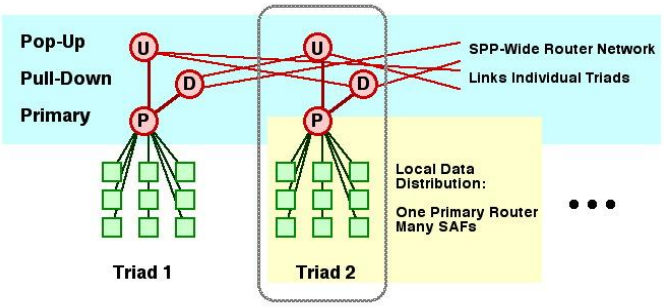

Figure 2 Schematic MeshRouter Topology

The overall communications scheme consists of collections of processors (labeled "SAFs" in this legacy diagram) each communicating with a specified "Primary" router (P). Interest-limited message exchange

[1] The naive run was killed at the elapsed time in the seventh job iteration. The maximum number for a run is 32.

among the various basic "Triads" is done using a network of additional "Pop-Up" and "Pull-Down" routers. As is described in [Barrett & Gottschalk 2004], the three routers on a triad are instanced as separate objects within a single MeshRouter process.

The execution of actual message transfer is implemented by a software stack as shown in Figure 4.

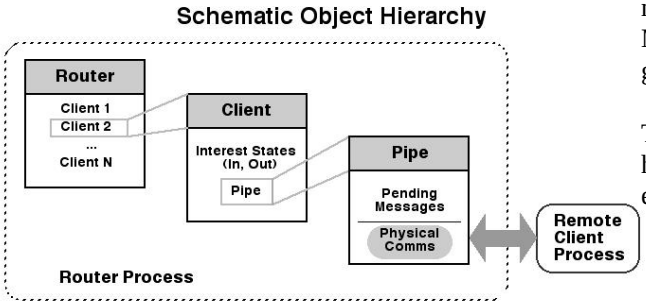

Figure 3 Factored MeshRouter implementation, with application-specific communications primitives.

The results reported here use an RTI-s implementation for both interest enumeration and the lowest-level communications primitives ("dataflow nodes"). While this has enormous advantages, it does have the generic disadvantage of any general purpose "plug and play" system in terms of significant, incompletely understood, overheads.

Standard RTI-s dataflow implementations exist for both TCP and UDP communications. The results presented here use the TCP implementation.

The application processes for the benchmark tests are of two forms:

Publish Processors: Send out messages of specified length and interest state. The nominal total publication rate (Mbyte/sec) is controlled by a data file that is re-read periodically (by all publish processors). This means that the nominal experimental data rate can be controlled dynamically.

Subscribe Processors: Receive messages for a specified interest state, collecting messages from multiple publishers, as appropriate. The subscribe processes are instrumented to measure actual incoming message rates and to detect missed messages.

The routers in Figure 3 direct individual messages from publishers to subscribers according to the interest declarations. The router processes are also instrumented to determine the fraction of (wall clock) time spend in communications management (versus simply waiting for input).

Two modes were tested. In the first mode, a single TCP connection was setup between a pair of meshrouters at distant locations. The measured bandwidth was approximately 300 megabits per second. The second mode used eight mesh routers at each site, each with multiple clients and multiple TCP connections. Measured aggregate bandwidth was approximately 4.6 gigabits per second.

This test demonstrated that 50% of the capacity of the high speed wide area network can be effectively employed by a real world application.

PROGRAMMING HADOOP

Hadoop was an easy system to use. Installation was straightforward. Although the rapid changes in Hadoop releases made keeping up problematic; some releases broke existing code. We did not install every updated release.

Shell scripts were needed to reduce the complexity of setup, change, and maintenance of the various Hadoop configurations across sites. Once these were in place, changing configurations was a quick operation.

Developing was convenient for Hadoop jobs. Running and debugging standalone Hadoop jobs in the Eclipse IDE allowed rapid turnaround on application bug fixes.

CONCLUSIONS

This paper reported on experiments using distributed data analysis/data mining implemented over the Apache Hadoop framework. We experienced that Hadoop provided a scalable, but conceptually simple, distributed computation paradigm based on map/reduce operations implemented over a highly parallel, distributed filesystem. We found it practical to develop map/reduce implementations of K-Means and Expectation-Maximization data mining algorithms that take advantage the Hadoop framework. The Hadoop filesystem dramatically improved the disk scan time needed by these iterative data mining algorithms. We successfully ran these algorithms across multiple Linux clusters over high speed networks which had been reserved. We hold that the results of these experiments point to potential enhancements for Hadoop and other analysis tools.

SUGGESTIONS FOR FUTURE WORK

It was our experience that the K-Means and EM Hadoop jobs are too tightly bound to a particular data set. Extending and generalizing these classes would make them amenable to a broader range of data. Hadoop's Map operation actually is made up of a sequence of finer grained stages. These stages include a file splitter that splits large files into smaller chunks, and a recorder reader that extract records from the chunks based on a given input format specification. We plan to make use of these preprocessing stages as a way to decouple the data parsing from the actual data mining algorithm.

The underlying network subsystem of Hadoop could be extended so that it allows Hadoop to take full advantage of the network resources. The high-bandwidth configurations used in the experiments slowed down Hadoop. Possible approaches to enhance the capabilities of Hadoop when deployed on wide-area, high-speed networks include:

- Adjusting how Hadoop utilizes the java.net and java.nio libraries such that, as with Meshrouters, Hadoop is able to treat one connection as a pipe with multiple TCP connections.
- Add support for the UDT protocol, a reliable transport protocol built on UDP [Yunhong 2007].

With these additional capabilities, Hadoop could more effectively support the data collection needs of complex, modern simulations

ACKNOWLEDGEMENTS

We would like to thank our colleagues at UIC, especially Professor Robert Grossman and the Messers Michael Sabala and Yuhong Gu. Further, this work would not have been possible without the assistance of Tom Lehman of ISI East. This material is based in part on research sponsored by the Air Force Research Laboratory under agreement number FA8750-05-2-0204. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

REFERENCES

Barrett, B. & Gottschalk, T. (2004). Advanced Message Routing for Scalable Distributed Simulations. In Proceedings of the Interservice / Industry Training, Simulation and Education Conference, Orlando, FL.

Brunett, S. & Gottschalk, T. (1998) A Large-scale Metacomputing Framework for the ModSAF Real-time Simulation . In Parallel Computing: V24:1873-1900.

Chu, C., & Kim, S., et. al. (2006). Map-Reduce for Machine Learning on Multicore. Retrieved June 15, 2009, from http://www.cs.stanford.edu/people/ang/papers/nips06-mapreducemulticore.pdf

Dean, J., Ghemawat S. (2004) MapReduce: Simplified Data Processing on Large Clusters. Operating System Design and Implementation. Retrieved 25 June 2009 from http://labs.google.com/papers/mapreduce.html

Graebener, R., Rafuse, G., Miller, R. & Yao, K-T. (2003) The Road to Successful Joint Experimentation Starts at the Data Collection Trail. In Proceedings of the Interservice/Industry Training, Simulation and Education Conference, Orlando, Florida.

Hadoop Map-Reduce Tutorial (2007) Retrieved 25 June 2009 from http://hadoop.apache.org/core/docs/r0.17.1/mapred_tutorial.pdf.

The Hadoop Distributed File System: Architecture and Design (2007) Retrieved 25 June 2009 from http://hadoop.apache.org/core/docs/r0.17.1/hdfs_design.pdf.

O'Malley, O. (2008) TeraByte Sort on Apache Hadoop. Retrieved 29 June 2009 at http://www.hpl.hp.com/hosted/sortbenchmark/YahooHadoop.pdf.

Powered By, 2009. Retrieved 29 June 2009 at http://wiki.apache.org/hadoop/PoweredBy.

Yao, KT., & Wagenbreth, G. (2005)Simulation Data Grid: Joint Experimentation Data Management and Analysis. In Proceedings of the Interservice/Industry Training, Simulation and Education Conference, Orlando, FL.

Yahoo! Launches World's Largest Hadoop Production Application, 2008. Retrieved 29 June 2009 from http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html.

Yunhong G., Grossman R. 2007. UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7.