Procedia Computer Science

Volume 51, 2015, Pages 1675–1684

ICCS 2015 International Conference On Computational Science

OpenDBDDAS Toolkit: Secure MapReduce and Hadoop-like Systems

Enrico Fabiano[1], Mookwon Seo[1], Xiaoban Wu[1], and Craig C. Douglas[1,2]

[1] University of Wyoming, Laramie, WY, U.S.A.
[2] King Abdullah University of Science & Technology, SRI-Center NumPor, Thuwal, K.S.A.
craig.c.douglas@gmail.com

Abstract
The OpenDBDDAS Toolkit is a software framework to provide support for more easily creating and expanding dynamic big data-driven application systems (DBDDAS) that are common in environmental systems, many engineering applications, disaster management, traffic management, and manufacturing. In this paper, we describe key features needed to implement a secure MapReduce and Hadoop-like system for high performance clusters that guarantees a certain level of privacy of data from other concurrent users of the system. We also provide examples of a secure MapReduce prototype and compare it to another high performance MapReduce, MR-MPI.

Keywords: Big data, DDDAS, dynamic data driven applications, HPC, open source software, cybersecurity, HIPAA

1 Introduction

Dynamic data-driven application systems (DDDAS) [1] is a paradigm that uses data assimilation to initiate a change in the models and/or scales of the computation and how the application controls the data collection based on the computational results. More precise predictions and more useful visualizations can be the result of applying such a technique to simulation applications.

Big Data is a paradigm for methods to data mine enormous amounts of data that is either streamed or contained in stored (and possibly growing) datasets. The field of computational sciences in recent years has merged with data intensive computing to become what is referenced as Big Data [2]. Big Data has become a superset of computational sciences with applications pervasive everywhere [3] leading to the interesting question, "What if you had all of the data?" [4]. The end result of combining Big Data and DDDAS (DBDDAS) is similar to the Data Intensive Scientific Discovery (DISD) [5], cyberphysical systems (CPS) [6] paradigms.

What amounts to a Big Data problem has changed over the last six decades. In addition, the smaller the time interval, the smaller the data size that transforms a problem into a Big

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.302

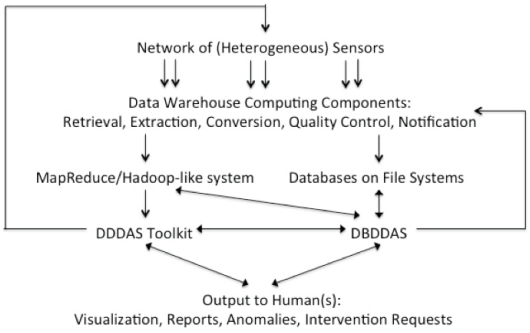

Figure 1: Typical use of OpenDBDDAS with a DDDAS

Data problem. What makes a problem Big Data is to a degree dependent on data throughput capabilities in a given situation. These issues are discussed in [7].

Most interesting DDDAS cases have Big Data aspects. These projects have discovered Big Data concepts with no field wide consensus on how to handle Big Data. Combining the two paradigms is crucial to creating new DBDDAS quickly and efficiently.

DBDDAS has already emerged in analysis, design, and business, engineering and scientific processes. Resource management, manufacturing process controls, traffic management, systems engineering, civil engineering, weather and climate prediction, geological exploration, social and behavioral modeling, cognitive measurement, and biosensing already benefit from DBDDAS.

The remainder of the paper is organized as follows. Section 2 is a summary of the OpenDB-DDAS Toolkit and its components. Section 3 describes a MapReduce system and Hadoop-like systems that are secure and guarantee privacy of data up to a certain level. Section 4 has examples. Section 5 has conclusions.

2 The OpenDBDDAS Toolkit

In this section, we describe a set of components that can be combined and used to construct a DBDDAS sufficiently conveniently that it will be usable by non-computer scientists. Similar projects that use this toolkit will have the advantage of being able to easily intermix pieces of their applications to construct better and unified ones. The cost of learning how to make an efficient DDDAS will also decrease. Figure 1 provides a flowchart of a typical usage of the toolkit.

The network of heterogeneous sensors provides the dynamic data to the application. The sensors can be traditional hardware sensors that just provide data. The sensors may be either dumb (just provide data at set intervals or on a query) or intelligent (can be reprogrammed at any time and have sufficient processing power to correct errors). The latter is referred to as an integrated sensing and processing (ISP) device. The sensor may be a reconfigurable software agent that interacts and collects data with data streams, databases, or web sites. We interpret the term sensor to be anything that can be used to intelligently collect data on demand.

The Data Warehouse is an integral part of the toolkit. It provides a flexible, but comprehensive interface to a set of databases that are hidden from the user. The functionalty provided is six-fold and are common in both Big Data and DDDAS.

1. *Retrieval*: Get the data from sensors, software agents, or databases. This may mean receiving data directly from a sensor or database or indirectly through another computer or storage device. There can be network side effects that cause data loss or damage.

2. *Extraction*: The data may be quite messy in its raw form. Thus, the relevant data may have to be extracted from the transmitted information.

3. *Conversion*: The units of the data may not be appropriate for all of the applications using the data warehouse. A consistent set of units is necessary.

4. *Quality control*: If possible, bad data should be removed or repaired and missing or incomplete data should be repaired.

5. *Store*: If the data needs to be archived, it must go to the right medium, which might be permanent or semi-permanent storage. If the data will not be archived, then it must exist only briefly and then be discarded if it is not used during a known and well defined time period.

6. *Notification*: Any simulation using the data must be informed as new data enters the data warehouse, which could necessitate either a cold or warm restart or a new start up.

ISP devices simplifies the data center entry process by performing data extraction and data conversion at the detector inside of the sensor, which eliminates steps 2 and 3 above. Data is delivered from the ISP as high level information tokens that require little to no extra communication bandwidth. Bad data may be edited, completed, or removed on the ISP as data are tokenized, potentially eliminating step 4, too.

The MapReduce and Hadoop-like systems will be discussed in more detail in Section 3. Unlike traditional such systems, the ones in the Toolkit are not based on disk files and are fast.

The DDDAS Toolkit consists of at a minimum the following components: (1) define DDDAS and its runtime requirements abstractly, (2) an open sensor description language and open source tools to use it, (3) data assimilation using multiscale interpolation methods and statistics based ensemble methods, (4) high performance disk I/O, (5) cache aware fast hashing table methods that do not use disk files, (6) uncertainty quantification and statistical analysis, (7) numerical solvers for standard nonlinear time dependent coupled PDEs and optimization, (8) fuzzy mathematics and logic, (9) item identification, tracking, and steering, (10) anomaly tracking and verification, and (11) human notification methods.

Many of these components already exist in some form or another. For example, NetCDF [8] and HDF5 [9] are robust parallel disk I/O systems in common usage.

Data assimilation is key to all DDDAS. Without it, the field would not exist. The use of it is quite application specific and the key research areas in DDDAS are different than those currently in the data assimilation field.

Hashing functions need to be much more sophisticated than the simple ones found in textbooks dating back to the 1960's. Memory cache aware functions can be created with little extra work and provide a useful speedup. Hashing is used in Big Data algorithms often. Very complicated hashing functions that are nontrivial and go beyond just hardware acceleration [10] can also be included in the Toolkit.

Many DBDDAS solve inverse problems to estimate semi-known parameters that mathematically define the models used. Small errors in the parameters lead to large errors in the predictions that come from the models. Quantifying the uncertainties and errors is an emerging field in science and very important to estimating the errors in the predictions [11].

A significant advantage of the DBDDAS paradigm is that the additional dynamic data from sensors and data sources during runtime leads to self correcting processes unavailable to traditional applications just based on static input datasets and vast numbers of samples to measure and quantify uncertainties that in the traditional case requires vast numbers of example runs to acquire far less data.

For any scientific or engineering based DBDDAS, solvers are required for working with the mathematical models. Linear, nonlinear, time stepping, and optimization method solvers must be available in the toolkit. Additionally, meshing and pre- and post-processing using graphical methods are also necessary in order to prepare initial inputs and to visualize predictions. None of these requirements need to be created anew since software libraries already exist. In the past few years many new libraries have been created to scale to Exascale computer systems that will exist later in this decade (e.g., [12]).

Fuzzy mathematics turns out to be important in many DDDAS since an interval, rather than a single number, is required. When dealing with statistical data and probabilities, fuzzy math and logic is a useful simplification to defining algorithms and their implementations.

Item identification, tracking, and steering is a large field of research with evolving results. Similarly, finding data anomalies is challenging [13]. An anomaly is defined as any data that is outside of the domain of expected data. Expected data is both application and time dependent within an application. A general purpose anomaly detection library is essential to the Toolkit and is still an active research area.

A notification library that provides methods to contact people or computer processes using network protocols, phone and voice methods, and texting must be a part of the Toolkit. It has to be easy to use and nonintrusive. In particular, the recipients of the library functions should not consider it a new spamming process.

Clearly, the Toolkit consists of complicated components that either already exist or are still being researched and constructed. To be effective and improve the likelihood of general adoption by application specialists, the Toolkit must be usable by non-computer scientists.

3 Secure MapReduce and Hadoop-like Systems

An open source Map-Reduce [14] suitable for medical records that must

1. Be HIPAA compliant at a minimum.
2. Scale well for very large databases.
3. Have individual user access capabilities.
4. Not have complexity O(disk access) on a distributed file system.
5. Detect and report inappropriate uses of the system.

Point (4) means that it should use OpenMP and MPI, not disk I/O.

Such a Map-Reduce system will be useful well beyond just medical records. For example, in energy based DBDDAS for running oil or gas reservoirs, many users will access a Hadoop-like system to monitor each reservoir. Reports need to be generated automatically based on how the reservoirs are used and changes in their management.

The Health Insurance Portability and Accountability Act of 1996 [15] provides a useful basis for defining how secure data needs to be while in multi-user MapReduce or Hadoop-like systems.

The remainder of this section is organized as follows. Section 3.1 describes a secure MapReduce system developed at the University of Wyoming. Section 3.2 describes user authentication. Section 3.3 describes how Map is implemented. Section 3.4 describes how Collate is implemented. Section 3.5 describes how Reduce is implemented.

3.1 UWMR: University of Wyoming MapReduce

In this section we describe UWMR, the MapReduce system implemented at the University of Wyoming. UWMR was developed to be HIPAA (the U.S. standard for medical record privacy) compliant [15] and makes extensive use of the OpenSSL encryption package [16]. MapReduce is a programming paradigm used to process large data sets on a large number of cores. It takes its name from its two main function. The user defined Map function filters and sorts the input data while the user defined Reduce function performs summary operations and presents results to the user. Although MapReduce has been developed by Google to perform web searches, its applications are virtually limitless and it represents the only viable tool to query significantly large data sets.

The current system was developed as part of a three months Big Data class at the University of Wyoming (Spring 2014). For this reason it cannot be considered a production-level code yet and further work is required. To achieve maximum scalability we made use of a hybrid OpenMP-MPI programming paradigm. Essentially each core is assigned a certain number of files to process and each core further subdivides all the assigned files among the number of threads available as shown in Figure 2.

The system works on two different sets of files. The first file is the "users file." It is a hidden, encrypted file that contains the username and password pair of all the users that have been granted access to UWMR. When a new user is granted access to our MapReduce system a third person, the system administrator must add the new username and password pair to the "users file." Hence the system administrator has access to the encryption keys and is also intended as a fail-safe system in case of a catastrophic failure of our system, as required by HIPAA federal regulations. The second set of files is the collection of files that our MapReduce will process once the user has been authenticated. Since UWMR has been primarily developed for medical applications we envision this set of file as a collection of medical records whose size is potentially significant, hence the need for a MapReduce system. However, as per HIPAA federal regulations, medical records can only be stored as long as the patient's privacy is comprehensively preserved. We comply with this requirement by storing medical records on disk as hidden files encrypted using OpenSSL.

The OpenSSL project provides software that is a full featured and robust implementation of the Secure Sockets Layer (SSL v2/v3) and transport layer security (TLS) protocols. It includes a general purpose, full strength cryptography library that we use extensively.

3.2 User Authentication

When the user attempts to log into UWMR, the system will first read, by means of elementary built-in C++ functions, the username and password pair provided by the user. Then it will decrypt to memory the "users file" and authenticate the user. If the login is unsuccessful the attempted login will be recorded in a log file, as per HIPAA federal regulations. If the login is successful, then MapReduce operations will continue, as per the user defined map function, as follows:

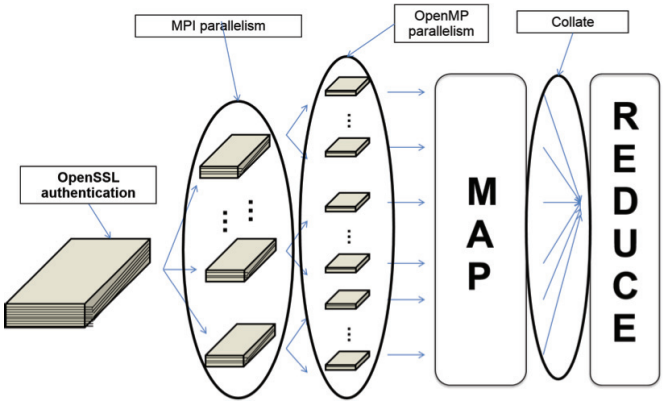

Figure 2: Workflow for UWMR and level or parallelism implemented

1. The root processor decrypts all of the assigned files to memory and creates several files that contain only the dataset that the user has access to.

2. MPI divides the total number of files in the dataset across all of the cores that the authenticated user has requested.

3. OpenMP divides the total number of files MPI has assigned to each processor by the number of threads available to each processor.

4. Starts the user's defined map function and generates the initial key-value pairs. Since the user's defined map function is built into the OpenMP environment, this function has to be thread safe.

Encryption and decryption are handled by the OpenSSL package [16]. OpenSSL is designed to provide communication security over a computer network. We rely on the Advanced Encryption Standard (AES) encryption specification, as it has been widely adopted by the U.S. Government and it has been deemed secure by the National Security Agency (NSA) as far as non-classified U.S. Government data are concerned, as are those we envision to protect in this work. The symmetric key algorithm used in this work has the standard 128 bits blocksize and a 256 bit key.

Currently the UWMR MapReduce system is a standalone program capable of handling only one user/job at a time. In order to turn UWMR into a fully capable Hadoop replacement the program should be able to manage different users operating it at the same time and have a managing system capable of executing the jobs and queue those whose requested resources exceed the cluster capability. Currently Hadoop requires a fully dedicated cluster under its control as it is incompatible with existing cluster batch job queuing systems. Furthermore the lack of functionality for job management and user access control makes Hadoop clusters difficult to maintain. To overcome these difficulties the Hadoop on Demand (HOD) application is usually added to Hadoop to make it more compatible with existing queuing systems. In practice HOD uses the existing cluster queuing system to schedule jobs and each jobs runs a Hadoop daemon. It has been noted that this approach does not take advantage of data locality. Oracle took a different path by integrating their Oracle Grid Engine SGE and Hadoop so that there is no need for a separate Hadoop dedicated cluster [17].

All of these issues can be easily overcome in UWMR. Since UWMR is a hybrid MPI-OpenMP parallelized program that does not make any use of an ad-hoc file system like Hadoop's HDFS it can easily be run on a shared cluster of nodes managed by a resource manager such as Torque or Moab. In fact during the development phase UWMR has been executed on the University of Wyoming Mount Moran cluster, which is managed by a Moab resource scheduler. The advantages of using either Torque or Moab are numerous. Job scheduling and resource allocation are already implemented.

An alternative to the use of an existing resource manager is to implement our own simple queuing system, which is what is found in Hadoop YARN. A list of desirable properties is listed in [17]. It is worth noticing that while no steps have been taken so far toward the implementation of a job queue management and a job scheduling and reservation system, UWMR already satisfies the need for user access control with the aforementioned OpenSSL user authentication.

3.3 Map

There are mainly three stages in the MapReduce system: map, collate and reduce. In the map stage, the user needs to call functions to generate and add the initial key-value pairs into the system. Each thread has a memory arena for all the keys and values. Each thread has an array of the structure that contains the detailed information of each key-value pair in that thread. The detailed information includes the pointer for the key, the size of the key, the pointer for the value and the size of the value. The pointer for the key points to some location in the KV_Key_Pool. The pointer for the value points to some location in the KV_Value_Pool. When the specific key-value pair is inserted, we need to store the key into the Key_Pool and store the value into the Value_Pool. We also need to store the information of the pointer for the key, the size of the key, the pointer for the value and the size of the value.

3.4 Collate

Each thread has a hash table (KMV_HT) of struct that contains the detailed information of each key-multivalue pair in that thread. The detailed information includes the pointer for the key, the size of the key, the pointer for the multivalue, the size of the multivalue, the number of values and the size of each value associated with this key. The pointer for the key points to some location in the KMV_Key_Pool. The pointer for the multivalue points to some location in the KMV_Multivalue_Pool. The purpose of this function is each processor gets all the key-value pairs it wants to handle among all the processors. It regroups all the key-value pairs by the hashing value of the keys. In order to fulfill this purpose, each processor needs to collect all the key-value pairs from each thread on this processor (KV_Key_Pool and KV_Value_Pool), puts them into the send-buffer and each processor needs to communicate with each other to get the necessary information to set up the recv-buffer. Next, MPI_Alltoall would finish the desired purpose. Finally, each thread adds the key-value pairs in the recv_buffer. We finish this function in two steps. In the first step, each thread inserts the desired key-value pairs among all the threads into the hash tale KMV_HT. The desired key-value pairs are those key-value pairs whose key hashing values are the same with the thread ID. During this insertion, each thread adds the key into the KMV_Key_Pool, stores the size of the key, stores the size of the multivalue and stores the number of values. After all these preparation work, each thread allocates the certain amount of the memory and go into the second step. In the second step, each thread inserts the desired key-value pairs among all the threads into the hash tale KMV_HT again. This time each thread adds the multivalue into the KMV_Multivalue_Pool and stores the size of each value associated with this key. The only concern is if we need to readjust the size of

Number of processes	Timing	
	MRMPI	UWMR
1	55.9856	15.996
2	48.3459	13.1709
4	31.6800	10.0048
8	25.9795	11.2008

Table 1: Timing for the word frequency example compared with MRMPI

the KMV_Key_Pool or the KMV_Value_Pool, since we have the preallocated certain amount of the memory. In order to do this, we add the the location of the corresponding struct into a queue to record the order during that insertion, so later if we need to readjust the size, we could update the pointer stored in the struct by following the recorded order.

3.5 Reduce

For development purposes it has been assumed that each file can be decrypted to memory. Should this not be possible because of the size of the file to be decrypted, paging techniques can be implemented where the file is split into separate parts, decrypting one part at a time and writing it to file. However, it should be noted that keeping all the data in memory and using message-passing interfaces is a unique feature of our UWMR system that allows it to be faster and more secure than a conventional MapReduce implementation where memory content is written to file only to be read again to memory later. Should writing memory content to file really be necessary we still guarantee security of the data by writing it to an encrypted files using again OpenSSL.

4 Examples

We tested UWMR on a Linux machine of 3.40 GHz 8 cores Intel Core i7-2600 CPU with 8 GB of RAM. We compared UWMR with MRMPI [18].

Our first example is the word frequency problem. The text file is 85.2 MB. The timing result is shown in Table 1. The output is a list of unique words, their count in the file, and the total number of words and unique words.

As a second example we compute the Mach number contours for the transonic flow around the NACA 0012 airfoil. This inviscid flow is simulated with an unstructured finite volume solver that achieves second order accuracy in space via matrix dissipation. A separate program reads the flow solution and for each cell of the computational mesh writes to file the local Mach number and the cell centroid coordinates. To plot the contours we used a binning technique. The user specifies the number of contours (bins) and the minimum and maximum mach number of interest. The user specified function computes the average Mach number for each contour (bin) which is the key for UWMR. UWMR then assigns each input Mach number to the proper contour (bin) and uses the associated centroid coordinate as the value. No reduce operation is needed and after sorting UWMR writes to file the contour value and the centroid coordinate. We use Tecplot's scatter plot capability to visualize the results and compare them to a Tecplot contour plot with data generated by the flow solver, as shown in Figure 3.

All of the flow features are correctly identified, with both the shocks being in the correct location, as well as the stagnation points. Small differences are present between the two plots

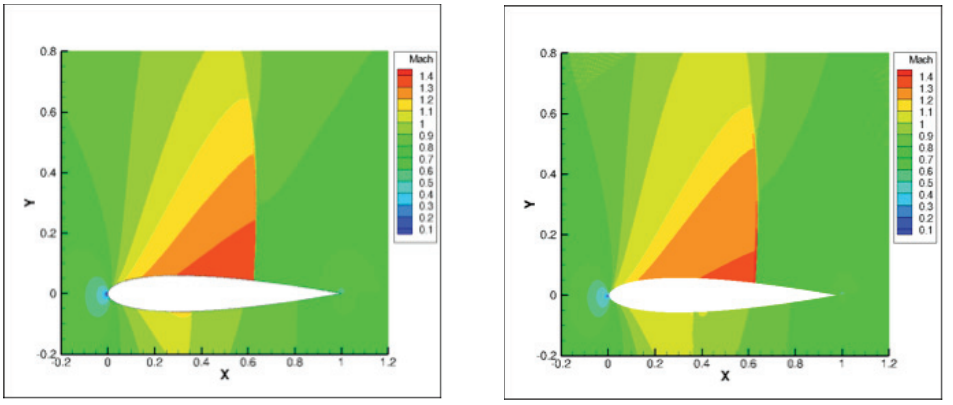

Figure 3: Compare two contour plots. Left figure is generated by Tecplot contour plot and right one is generated by Tecplot scatter plot with UWMR output data

due to the different contour level definition between Tecplot and UWMR.

5 Conclusions

The OpenDBDDAS Toolkit should have at a minimum, the following components: a general definition scheme for a DDDAS and its runtime requirements, uncertainty quantification and statistical analysis, numerical solvers for standard (nonlinear time dependent coupled) PDEs, optimization solvers, data assimilation using multiscale interpolation methods, item identification, tracking, and steering, anomaly tracking and verification, and human notification methods. Additionally, data can be better analyzed, visualized, managed, and accessed faster using the Big Data paradigm than ad hoc data management methods created by nonexperts who excel otherwise in application simulators.

Finally, we have described a prototype of an Open Source, secure MapReduce system and how to convert it into a fully functional Hadoop-like system with multiple users, queuing, and scheduling. We use as a basis the HIPAA requirements to provide privacy to the data at runtime. Further, our MapReduce systems uses a hybrid OpenMP-MPI system for parallel communication that is both fast and scalable. Encryption is provided using the Open Source OpenSSL library. Examples have also shown that the prototype is already faster than one of the high performance computing versions of MapReduce.

As a result of the performance of the prototype, we have begun a production version of UWMR that will be made available when it is completed. Each part of the new UWMR is being carefully tuned for speed. A notable feature is that it does not use the disk drives as intermediate data holders. Hence, it does suffer from terrible performance as most MapReduce systems do.

Acknowledgments

This research was supported in part by National Science Foundation grants EPS-1135483 and ACI-1440610.

References

[1] DDDAS.org. http://www.dddas.org, 2001-2015. Last visited: February 5, 2015.

[2] RE Bryant, RH Katz, and ED Lazowska. Big-data computing: Creating revolutionary breakthroughs in commerce, science, and society, ver. 8, 2008.

[3] T Hey, S Tansley, and K Tolle. The fourth paradigm: Data-intensive scientific discovery, 2009.

[4] J Manyika, M Chui, B Brown, J Bughin, R Dobbs, C Roxburgh, and AH Byers. Big data: The next frontier for innovation, competition, and productivity, 2011.

[5] T Hey, S Tansley, and K Tolle. The fourth paradigm: Data intensive scientific discovery, 2009.

[6] Cyberphysical systems. http://cyberphysicalsystems.org. Last visited: February 5, 2015.

[7] CC Douglas. An open framework for dynamic big-data-driven application systems (dbddas) development. *Procedia Computer Science*, 29:1246–1255, 2014.

[8] R Rew, G Davis, S Emmerson, H Davies, H Hartnett, D Heimbigner, and W Fisher. Netcdf, 1989-2015. http://www.unidata.ucar.edu/software/netcdf. Last visited: February 5, 2015.

[9] HDF Group. Hierarchical data format: Hdf5, 2008-2015. http://www.hdfgroup.org. Last visited: February 5, 2015.

[10] R Jenkins. A hash function for hash table lookup, 2001. http://burtleburtle.net/bob/hash/doobs.html. Last visited: February 5, 2015.

[11] L et al Biegler. *Large-Scale Inverse Problems and Quantification of Uncertainty*. John Wiley & Sons, Chichester, 2011.

[12] J Kurzak, P Luszczek, A YarKhan, M Faverge, J Langou, H Bouwmeester, and J Dongarra. *Multithreading in the PLASMA library, Multi and Many-Core Processing: Architecture, Programming, Algorithms, & Applications*. Taylor & Francis, 2013.

[13] V Chandola, A Banerjee, and V Kumar. Anomaly detection: a survey. *ACM Computing Surveys*, 41/3, article 15:1–58, 2009.

[14] J Dean and S Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004*, pages 137–150, 2004.

[15] U.S. Government. Health insurance portability and accountability act, 1996. http://www.hhs.gov/ocr/privacy/index.html. Last visited: February 5, 2015.

[16] OpenSSL. http://www.openssl.org, 1999-2014. Last visited: February 5, 2015.

[17] C Zhang and H De Sterck. Cloudbatch: A batch job queuing system on clouds with hadoop and hbase. In J Qiu, G Zhao, and C Rong, editors, *CloudCom 2010: 2nd IEEE international Conference on Cloud Computing Technology and Science*, pages 368–375, Los Alamitos, CA, 2010. IEEE Computer Society.

[18] S Plimpton and K Devine. Mapreduce-mpi library (mr-mpi), 2009-2010. Sandia National Laboratories, http:mapreduce.sandia.gov. Last visited: March 31, 2015.