YARNsim: Simulating Hadoop YARN

Ning Liu, Xi Yang, Xian-He Sun
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois
{nliu8, xyang34}@hawk.iit.edu, sun@iit.edu

Johnathan Jenkins, Robert Ross
Mathematics and Computer Science Department
Argonne National Laboratory
Argonne, Illinois
{jenkins, rross}@mcs.anl.gov

***Abstract*—Despite the popularity of the Apache Hadoop system, its success has been limited by issues such as single points of failure, centralized job/task management, and lack of support for programming models other than MapReduce. The next generation of Hadoop, Apache Hadoop YARN, is designed to address these issues. In this paper, we propose YARNsim, a simulation system for Hadoop YARN. YARNsim is based on parallel discrete event simulation and provides protocol-level accuracy in simulating key components of YARN. YARNsim provides a virtual platform on which system architects can evaluate the design and implementation of Hadoop YARN systems. Also, application developers can tune job performance and understand the tradeoffs between different configurations, and Hadoop YARN system vendors can evaluate system efficiency under limited budgets.**

To demonstrate the validity of YARNsim, we use it to model two real systems and compare the experimental results from YARNsim and the real systems. The experiments include standard Hadoop benchmarks, synthetic workloads, and a bioinformatics application. The results show that the error rate is within 10% for the majority of test cases. The experiments prove that YARNsim can provide what-if analysis for system designers in a timely manner and at minimal cost compared with testing and evaluating on a real system.

Keywords: Hadoop; MapReduce; YARN; Parallel Discrete Event Simulation; HDFS

I. INTRODUCTION

Apache Hadoop [1] is the most popular open-source implementation of the MapReduce [2] framework. However, the first generation of Hadoop is based on a centralized model, where a JobTracker manages both resource allocation and job processing. This design is convenient for implementation and management, but its inherent limitations become issues as Hadoop enters an era of jobs with millions of concurrent tasks and systems with tens of thousands of nodes. The master JobTracker is a single point of failure, meaning that a simple failure in the JobTracker will bring down the entire system. It is also a bottleneck for scalability, since all jobs and tasks are managed through this single point. In HDFS [3], the NameNode functions similar to the JobTracker. It manages all files, the associated blocks, and their locations.

To best quantify such design limitations and aid in the design and implementation of Hadoop systems, many researchers have leveraged tools such as simulators or emulators. Quite a few Hadoop performance models and simulation systems exist. In terms of simulation techniques used, Mumak [4], SimMR [5], MRperf [6], MRSG [7], and MRsim [8] are based on discrete-event modeling and simulation, and Starfish [9] [10] is based on analytical modeling and simulation. In terms of system design, Mumak and SLS [11] are Hadoop scheduler simulators focusing on MapReduce job scheduling and resource allocation; and MRperf, MRSG, and MRsim focus on simulating the MapReduce job/task execution engine.

Simulations have provided useful insights into the Hadoop system performance. Most important, simulation provides a timely and cost-effective way to design and validate a next generation system. However, these simulation systems are constrained because (1) most discrete-event simulation systems are based on a sequential event processing engine, limiting the fidelity of simulations that may be run in a reasonable time; (2) simulation systems based on analytical models are unable to accurately model complex systems, and the accuracy is often sacrificed because of the necessary simplifications [12]; and (3) no a comprehensive simulation system exists for modeling Hadoop that includes both the software stack and hardware components. For example, Mumak and SimMR cannot capture the resource contention because of the lack of detailed network models. MRperf is based on the network simulation system ns-2 [13] but lacks a detailed model for HDFS and local file systems. In fact, most simulation systems do not include a model for HDFS.

Hadoop YARN [14] was recently proposed as the next-generation Hadoop system, addressing issues in the first-generation and providing new capabilities. The central idea of Hadoop YARN is to provide a generalized platform for managing resources and supporting diverse programming models, including MapReduce and MPI. Many researchers have already focused on the performance optimization of Hadoop YARN [15]. However, the community still lacks a comprehensive Hadoop YARN simulation system that can provide a sandbox for exploring design alternatives and understanding YARN behavior. In this paper, we present YARNsim: a simulation system for Hadoop YARN. Our contributions are as follows:

- To the best of our knowledge, YARNsim is the first comprehensive Hadoop YARN simulation system.
- YARNsim runs in parallel and thus is capable of modeling and simulating large-scale scenarios.

- YARNsim includes a detailed, protocol-level accurate HDFS model that increases simulation fidelity. A detailed HDFS model is not covered in most existing MapReduce simulation systems.
- We develop a bioinformatics application model and verify its accuracy through comparing performance results from both YARNsim and a real system.

The rest of the paper is organized as follows. Section II addresses the motivation and background of YARNsim. Section III presents the design and implementation of YARNsim system components, including a MapReduce module, an HDFS module, and a local storage system module. Section IV discusses the experimental evaluation and demonstrates the validity of YARNsim. Section V introduces the related works. Section VI summarizes our conclusions and points out future directions of this study.

II. Motivation and Background

In this section, we first present the motivation of developing YARNsim from two perspectives. We then discuss the related systems used by YARNsim.

A. Big Data

In the past few years, the focus of large-scale parallel and distributed processing has transitioned gradually from computation to storage. Increasing compute capabilities have generated enlarged data-processing needs, encouraging scientists in different endeavors to pursue bigger and more complex problems. However, the capability of the storage system has severely lagged behind in terms of both access speed and capacity. Even with the wide use of enhanced storage media, such as SSD, and countless data caching and data placement optimization techniques [16], [17], the gap between the capabilities of computation and storage systems still increases. As suggested in [18], the gap will continue to widen given the current technology trends. Scientific applications are severely constrained by the I/O bottleneck in data preloading, defensive checkpointing and simulation postprocessing. Commercial applications face similar dilemmas, where users are generating data at an unprecedented rate and the data processing and storage needs quickly go beyond the capacity of current storage system. For example, Facebook users were storing a total of 300 petabytes of data as of November 2013 [19].

The challenge of big data, at its core, is large-volume data processing, movement, and storage. Thus, a successful Hadoop simulation system should also include the model for data and its related operations to cater to the challenges of big data. In particular, YARNsim differs from other MapReduce simulators in that it provides more accurate models describing details of data movement, communication contention over the network, and task interactions due to data request conflicts.

B. Hadoop

Because of its simplicity and efficacy, MapReduce is one of the most well-accepted programming models in the distributed-computing community. According to a report from Gartner [20], 65% of the packaged data analytic applications will be built on Hadoop by 2015. Because of Hadoop's popularity, we are motivated to develop a tool that helps the community better understand the advantages and disadvantages of Hadoop systems and quantitatively measure the trade-offs between different design points.

Good scalability is one of the primary reasons Hadoop is widely adopted for large-scale data analytic solutions [21]. As the system grows in size, scalability has also become a top challenge facing Hadoop development [22]. Although Hadoop YARN has separated the management of resources and jobs and largely decreased the load of the master, the resource management and scheduling are still based on centralized model and will face scalability issues again as the need for computation and data movement keeps increasing. As Hadoop YARN proposes to support diverse programming models (in addition to MapReduce), the system may encounter new challenges in terms of system service design, fair resource scheduling, and system utilization in a hybrid workload environment. It is favorable to have an efficient simulation to aid in the evaluation and redesign of a complex system such as Hadoop YARN after deployment, especially since different datacenters may favor different domains of applications and since system architectures (in terms of both software and hardware) varies considerably from one to another. Thus, an efficient and cost-effective tool is necessary and the key to successful system design. In this paper, we show that YARNsim can be used to evaluate MapReduce jobs at the minimal cost of mainly CPU cycles.

C. Background

YARNsim is based on two parallel simulation packages: the Rensselaer Optimistic Simulation System (ROSS) [23] and the Co-Design of Exascale Storage System (CODES) [24].

ROSS is a massively parallel discrete-event simulator that has demonstrated the ability to process billions of events per second by leveraging large-scale HPC systems [25]–[27]. A parallel discrete-event simulation (PDES) system consists of a collection of logical processes (LPs) each modeling a distinct component of the system (e.g., a DataNode). LPs communicate by exchanging time stamped event messages (e.g., denoting the arrival of a new I/O request at that node). The goal of PDES is to efficiently process all events in a global timestamp order while minimizing any processor synchronization overheads. Two well-established approaches toward this goal are broadly called conservative processing and optimistic processing. ROSS supports both approaches. In this paper, we adopt the conservative approach.

CODES is a simulation system based on ROSS. Its goal is to enable the exploration and co-design of exascale storage systems by providing a detailed, accurate, and highly parallel simulation toolkit for exascale storage systems. Two particularly important modules in CODES are the network module (CODES-net) and the local storage model (CODES-lsm). Specifically, CODES-net provides parallel discrete-event-based networking models. The current CODES-net toolkit includes four modules: a torus network model [26], a dragonfly network model [28], a loggp [29] model, and a simple bandwidth-latency model called "simple-net." CODES-net provides a unified user interface that facilitates the use of

all these underlying networking models. For example, Tang et al. has buit a cloud scheduler for multi-workflow system [30].

III. YARNSIM DESIGN AND IMPLEMENTATION

The Hadoop system is inherently complex. In particular, its design features a multilayered and multicomponent software architecture. The underlying hardware model is flexible, in that a variety of low-end commodity-level equipment could be deployed, and thus adds complexity to the design space. The community needs the ability to understand and redesign such complex systems with limited time and budget. One viable approach is to use parallel discrete-event simulation. A successful simulation system design relies on accurate abstraction of the software and hardware stacks and pinpointing the key components from the big picture.

TABLE I. MODELED SYSTEM PROCESSES

Daemon Process	YARNsim Abbreviation
Client	CLT
Resource Manager	RM
Application Manager	AM
Resource Scheduler	RS
Node Manager	NM
Map Task	MAP
Reduce Task	RED
Application Master	AMS
Resource Scheduler	RS
Node Manager	NM
Map Task	MAP
Reduce Task	RED
Application Master	AMS
CODES-lsm	LSM
CODES-net	CN
NameNode	NN
DataNode	DN

Table 1 list all the Hadoop modules modeled in YARNsim from a daemon process perspective. The abbreviations are used throughout the rest of the paper.

Hadoop YARN and its ecosystem comprise millions of lines of source code. One of our targets in designing YARNsim is to develop a rich simulation that captures the salient properties of this complex system so as to mimic the behavior in terms of task execution and data flow. In YARNsim, we categorized the internal communication to metadata flow and data flow. Both flows can have a significant impact on the overall system performance. The model of metadata flow follows the basic protocols used in YARN. Specifically, YARN internal communication follows the client-server model and is based on a customized RPC library. A Hadoop RPC server consists of a listener module and a callQueue module. The message passing between different daemon processes is essentially RPC calls. We follow the client-server model in the design of YARNsim's communication system. Specifically, a client and a server are modeled as two distinct LPs, and the message passed between the client and server are modeled as events sent and received between LPs. Our model does not include thread-level details, but the server models the message queuing effects. We later show that this level of abstraction provides a reasonable accuracy in performance evaluation.

The YARN design is also based on event-driven and state-machine modeling. Specifically, a state transition is defined by a tuple <preState, postState, event, hook>, where the preState and postState are the actual states of the system components (e.g., resource manager), the event is the system-defined transition trigger (e.g., a task to get the requested resource is a transition trigger), and the hook is the execution function corresponding to each event. Similarly, YARNsim uses these concepts in the model and simulation. The changes of states and event communications are the exact flow logic we want to capture in YARN. Here, we made many simplifications and abstractions. For example, we don't model the security modules in the current version and leave this for future development. We don't differentiate between preState and postState; that is, the state transition is regulated by the transition function, which is based on the RPC protocols. We also merge events if they are functionally similar and will not affect performance.

Hardware is a key factor affecting the Hadoop system and application performance. In general, the hardware models considered in YARNsim are CPU, memory, network, and disk. We use analytical models for CPU and memory, since they are ideal in modeling less complex scenarios in terms of data movement.

$$T_{\mathrm{m}} = \frac{L_{\mathrm{data}}}{B_{\mathrm{m}}} + t_{\mathrm{m}} \tag{1}$$

$$T_{\mathrm{cpu}} = \alpha_{\mathrm{a}} \cdot \frac{L_{\mathrm{data}}}{P_{\mathrm{cpu}}} + t_{\mathrm{cpu}} \tag{2}$$

Equations 1 and 2 describe the analytical model used in YARNsim to compute the data movement cost in memory and the data processing cost in CPU. In Equation 1, we define the data transmission time (T_{m}) such that it is influenced by data request size (L_{data}), memory bandwith (B_{m}), and memory access latency (t_{m}). In Equation 2, we define the data-processing time (T_{cpu}) such that it is determined by data request size (L_{data}), CPU speed (P_{cpu}), CPU context switch time (t_{cpu}), and request intensity (α_{a}). Here, request intensity α_{a} is a factor used to decide whether a request is compute intensive or data intensive and thus is application dependent. We argue that these simplified models are sufficient to model the data-flow latency in a single node and are on a par with other components of YARNsim.

The I/O system is usually more complex than the CPU and memory system in terms of data-processing latency. I/O latency is determined by many factors, including request size, request pattern (continuous or noncontinuous), request type (read or write), and media type (HDD or SSD). It is less compelling to use a linear analytical model to describe such a nonlinear system. Many other MapReduce simulation systems such as MRperf [6] and MRsim [8] don't have a comprehensive I/O system model. We argue that the I/O system can have a significant impact on the overall system and application performance. We use CODES-lsm to model a disk for a single physical node. CODES-lsm is a coarse-grained hardware model that captures the nonlinear nature of HDD in response to different I/O request offsets and sizes. Before

configuring the YARNsim disk model, users are required to collect data about the I/O performance and feed it to the disk model such that the simulation system can return the correct I/O latency when

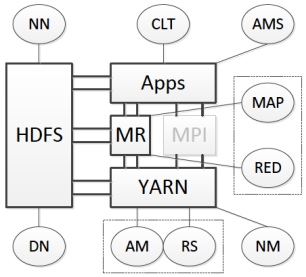

Figure 1. YARNsim architecture

provided with a real or synthetic request. Current experimental results show that YARNsim disk model is suitable for our simulation purposes. If a higher-fidelity disk model is required, one can extend this model or replace the CODES-lsm module and still use the standard interface.

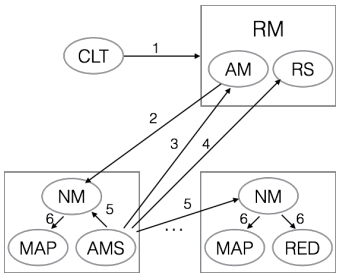

Figure 2. YARNsim workflow

The hardware-related parameters are configurable through user-defined input configuration files. Upon initialization, users can customize the Hadoop hardware system and get an accurate hardware platform. The software stack model of YARNsim is illustrated in Fig. 1. The major components are the application module, MapReduce module, YARN module, and HDFS module. The ovals are used to illustrate the daemon processes modeled in YARNsim. Each oval represents one type of daemon process. We describe in detail each LP and its functionality in the following sections.

A. Application Module

YARNsim uses one type of LP to model a client, and it is a key component of the application Module (Apps). A client LP is responsible for collecting application information and sending it to RM upon YARNsim initialization. At the end of the simulation, CLT collects simulation statistics and provides the user with an estimation of application performance. We currently provide job parameters in the input configuration file. Thus the client LP will load these parameters at initialization. In the case of multiple clients, there are multiple LPs, and each is responsible for initializing a specific application and collecting the corresponding information after the simulation terminates.

B. YARN Module

The workflow of YARN is illustrated in Fig. 2. Here, the rectangle boxes represent a modeled physical node, and ovals represent the daemon processes modeled in YARNsim. The numbered labels in Fig. 2 represent the order of the data flow in YARNsim. For example, a CLT initially submits a job to AM and gets an acknowledgment. Internally, AM updates the global job queue and handles the job execution. YARNsim uses these flows to model the protocols used in YARN. In Fig. 2, the RM consists of two services provided separately by application manager and resource scheduler. The two services are functionally independent but are all designed as a centralized communication point. We use two different types of LPs to model them because they are the potential bottleneck in the simulation system. The AM LP handles job submission and processing, while the RS LP implements the scheduling algorithms. Currently, YARNsim supports the default FIFO scheduling algorithm.

Application Manager LP

```
1: /* accept job submission event */
2: job_queue ← submitted new jobs
3: ack back job submission
3: while job_queue non empty do
4:    execute jobs
5: end while

1: /* job execution event */
2: inquire node manager LP
3: if container available then
4:    init AMS
5:    acknowledge AMS
6: else
7:    job_queue ← current_job
8:    execute jobs
9: end if

1: /* heart beat event */
2: if heart_beat_interval > threshhold then
3:    register dead_node
4:    update resource_lists
5:    reallocate AMS
6: else
7:    claim released resources
8: end if
```

Figure 3. Selected event algorithms used in application manager LP

Compared with real system implementations, YARNsim makes a number of simplifications. For example, the container manager is the key module in the node manager daemon. It contains many services, including the resource localization service, container launcher, auxiliary services, container monitor, log handler, and container event dispatcher. We do not model all the services but, instead, focus on the data-related services and abstract these to events flowing in and between the LPs. For example, in Fig. 3, we list the three selected

events and the core algorithms used in the application manager LP. The accept-job event manages the centralized job queue and schedules the job execution and termination. The job-execution event handles the execution of a job in YARNsim, and the heartbeat event models the resource dynamic allocation and reallocation related to each heartbeat message.

Node Manager LP

```
1: /* AMS request event */
2: check resources_list
3: if container available then
4:     allocate resource
5:     update resources_list
6: else
7:     return exception
8: end if

1: /* AMS release event */
2: collect released resources to resource_list
3: ack back AMS
4: update NM info to RS
```

Figure 4. Selected event algorithms used in node manager LP

In Figs. 4 and 5, we list the selected events and the core algorithms used in NM and RS, respectively. Here, each NM keeps a local list of resources. When AMS queries NM for a container for a specific task, NM will make decisions based on the resource list. This procedure is the AMS request event. AMS controls the job and tasks execution and will release a specific container when the corresponding task is finished. This procedure is the AMS release event. The RS basically implements the scheduling module. Currently, YARNsim supports FIFO scheduling; we plan to incorporate fair and capacity schedulers in the near future.

Resource Scheduler LP

```
1: /* application request event */
2: check resources_list
3: init scheduling module(request)
4: if schedule successful then
5:     grant resources
6:     update resources_list
7: else
8:     ack deny message
9: end if
```

Figure 5. Selected event algorithms used in resource scheduler LP

C. MapReduce Module

YARN is designed to support multiple programming models, including MapReduce. In YARNsim, the MapReduce module is a model of the first-generation Hadoop system, except for a few major differences including the model for YARN application master, which is responsible for task management. When a job is submitted and the corresponding application master gets the allocated container, it can initiate the MapReduce job. In YARNsim, the map task and reduce task are modeled as different types of LPs, respectively. In Fig. 6, we illustrate the map task model. Here, the HDFS model, the local storage system model, and the network model are involved. The HDFS model is discussed in detail in the following section. We use CODES-lsm module to model the local file system and the corresponding hardware. To capture the shuffle phase network contention, we use CODES-net as the underlying network abstraction. CODES-net provides protocol level accuracy in modeling network behaviors. CN LP is the network interfaces where a task LP can send and receive messages.

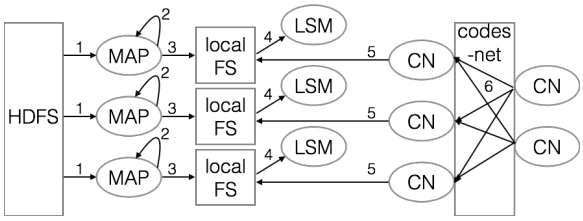

Figure 6. Map task model

The centralized resource scheduler allocates containers to each job and task. Upon initialization, the map task starts reading the input split from HDFS; this procedure is labeled as Flow 1 in Fig. 6. Here, we illustrate the YARNsim MapReduce module through an example MapReduce job in which the number of map tasks is three and the number of reduce tasks is two. The map task computation time is given in Equation 2. The intermediate results are saved to the local file system; this procedure is labeled as Flow 4. We use a configurable parameter slow.start.rate to simulate the start of a reduce task. This is the same as defining the slow start rate in a real system. The reduce tasks are based on a pull model, and thus Flow 5 is a set of requests sent by the reduce task to retrieve data from the map tasks. The circular buffer is modeled by using Equation 1, where a local file commit is triggered if the buffer is filled. In the shuffle phase, the reduce task can retrieve data from memory or local file system. Thus performance depends on the application.

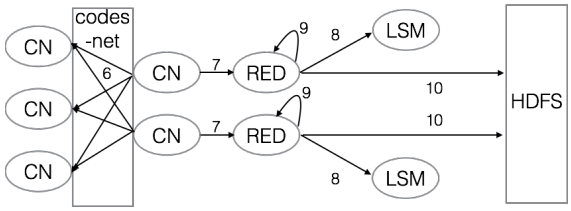

Figure 7. Reduce task model

The reduce task model is illustrated in Fig. 7. The computation time is based on Equation 2, and the I/O time is based on the local file system and HDFS models. As we can see, YARNsim models all key factors contributing to the overall job execution time. Any change in the system could lead to variation in the system/application performance. System designers can leverage the functionalities provided by YARNsim and choose to focus on a specific component to study the best optimization solutions.

D. HDFS Module

As discussed, YARNsim includes an HDFS module. The network data transmission in the HDFS module leverages the features provided by CODES-net. We illustrate its details in Fig. 8. Here, the NameNode daemon and data daemon are modeled as two types of LPs. The client daemon is the HDFS client, which could be a map task or a reduce task. For example, upon initialization, map tasks start reading input split from HDFS. In YARNsim, the map task LP starts by contacting the DataNode LP and specifying its I/O request. The DataNode LP forwards this request to the underlying local storage model LP that gives the exact delay for such request. When a map task tries to load a remote data block, the request goes through the network. In YARNsim, the request goes through the CODES-net module, and therefore we can capture the network contention and accurately account for the delay in retrieving the data block.

In the MapReduce programming model, the NameNode keeps an image of the HDFS file system. The image contains information about data block locations, IDs, file association, and all the replicas. In YARNsim, we leverage the libxml library to build a similar but simplified HDFS image and store it in the NameNode LP. We use this image for HDFS model management and operations. For example, if an application writes data block to HDFS, this image is updated with new data block information. The NameNode LP also tracks the status of DataNode LPs through heartbeat events. Similar to other components, we made several simplifications. For example, we do not yet support the secondary NameNode, and each DataNode LP does not keep the real data block but rather the metadata about the block. Currently, the HDFS module does not support DataNode additions or removals, contrary to real systems. We argue that this kind of simplification is sufficient for system modeling and evaluation. As YARNsim grows, one can extend the support to include the above-mentioned features.

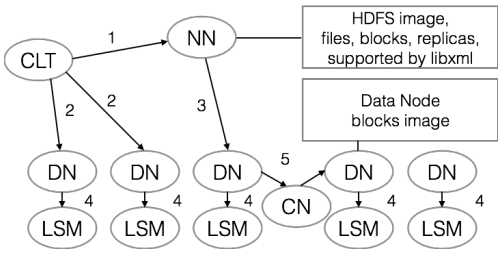

Figure 8. HDFS model

Figures 6–8 provide intuitive examples to explain the details of YARNsim. In reality, the number of different type of LPs and the corresponding parameters is configurable through user-defined input files. A YARNsim user can configure a customized YARN system of arbitrary size and run a customized application for evaluation and validation.

IV. Experimental Evaluation

Our experiments are conducted on two small-scale clusters, Craysun and HEC, in the Scalabe Computing Software Laboratory at the Illinois Institute of Technology. Craysun has 16 nodes, each node equipped with a quad-core Intel Xeon CPU geared at 3.40 GHz. However, each node currently only has 1 GB of memory. For this study, we want to mimic the situation where a MapReduce framework is built on a low-end cluster with commodity-level devices. HEC is a 51-node Sun Fire Linux-based cluster having one head node and 50 computing nodes. The head node is a Sun Fire X4240, equipped with dual 2.7 GHz Opteron quad-core processors, 8 GB memory, and 12 500 GB 7200RPM SATA-II drives configured as a RAID5 disk array. The computing nodes are Sun Fire X2200 servers, each node with dual 2.3GHz Opteron quad-core processors, 8 GB memory, and a 250 GB 7200RPM SATA hard drive. All 51 nodes are connected through Gigabit Ethernet. As of this paper, the most up-to-date Hadoop YARN is version 2.5.0, which we used for all our experiments.

A. YARNsim Configuration

YARNsim is a comprehensive simulation system with both software and hardware models. Thus, proper configurtion of the system is important for obtaining faithful simulation results. YARNsim uses CODES-lsm as the storage system model where its input is based on a set of parameters indicating varied read/write performance for different request sizes. We use the IOzone [31] benchmark to get the disk performance curve for both Craysun and HEC. Only the results on HEC are presented in Fig. 9 because of limited space. Here, we tested both read and write performance under both random and sequential I/O modes. The reported results are the average of 10 identical runs. In HDFS, we assumed that the data blocks are transferred sequentially between file system clients and servers. Thus the points in sequential mode were used as input to CODES-lsm. YARNsim's local file system component deals with the merge-sort I/O requests in map and reduce tasks, so we assumed that those data are random, and we used the random I/O performance as input. As Fig. 9 indicates, the curve depicts the nonlinear nature of HDD performance under varied I/O requests. This curve was used as a real disk performance in YARNsim and thus provides a trustworthy hardware model as compared with the fixed bandwidth analytical model used in other MapReduce models[6][9].

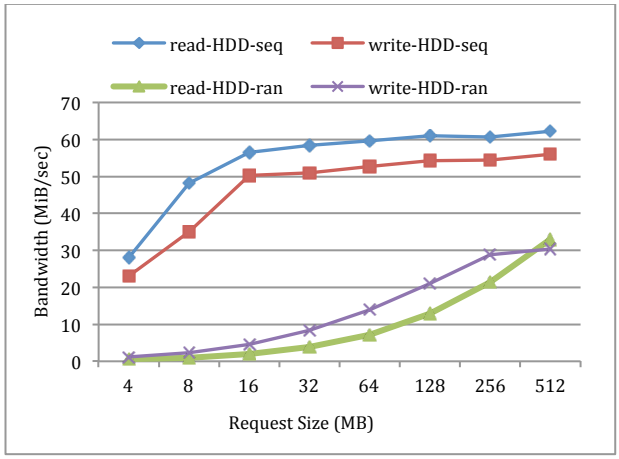

Figure 9. IOzone benchmark tests on HEC

In YARNsim, the HDFS model records only the simplified metadata information, namely the files, blocks, and their locations. Before a MapReduce job starts, the name of the

required data file is presented to NM, and NM will respond with the corresponding block location information. The scheduler will then schedule the corresponding tasks. This input determines the balance and data skew of a specific MapReduce job and thus the overall job execution time.

YARNsim uses the CODES-net module as the network layer simulator. In this study, we used the simple-net module, which models the switch connected Ethernet topology. It is a loosely defined p2p network with full connections at each node. As we show later, this module best suits our needs in modeling the network topology of Craysun and HEC.

Similar to running a real job in Hadoop YARN, both benchmark jobs and application jobs can be configured before execution. Users choose to define the number of tasks used, the running mode, and so forth. In YARNsim, users are required to provide a little more information to help YARNsim accurately capture the characteristics of each job. For example, users must tell YARNsim if the submitted job is computation intensive or data intensive through a user-defined configuration file. The weight is usually job-specific. Benchmarks such as TestDFSIO and Teragen are usually considered data intensive, whereas K-means and grep can be considered compute intensive.

Figure 10. Performance comparison between HEC and YARNsim on Teragen benchmark: input data size varies from 128 MB to 16 GB; the number of nodes is 16.

B. Hadoop I/O Benchmarks

We chose Teragen as the I/O benchmark tests to focus on the HDFS model validation. Also, the generated data were used as inputs for the following benchmark tests in the real system. In Craysun, we used a total of 16 nodes and varied the generated data from 128 MB to 16 GB. The block size was configured as 32 MB, the number of map tasks was configured to match the number of blocks, and the number of reduce tasks was the default value 1. In YARNsim, we also configured the system as above. To compare the results from Craysun and YARNsim, we built a model for Teragen in YARNsim to simulate the data generation process. The comparison results are reported in Fig. 10. The errors between simulation results and real system results are with 10% for all test cases. Since the data is balanced out to each compute node, we can get a balanced MapReduce tasks execution in the following tests.

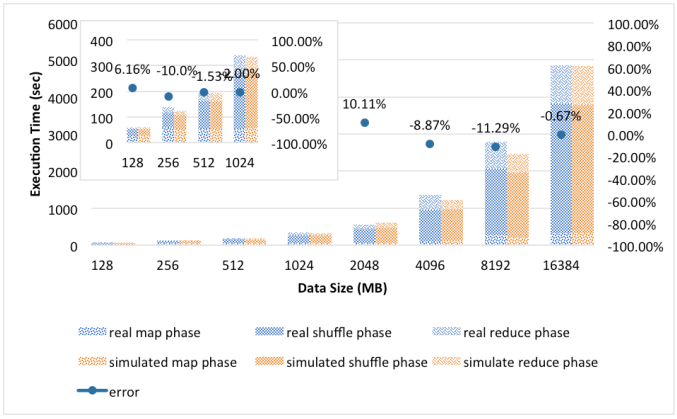

Figure 11. Performance comparison between Craysun and YARNsim on Terasort benchmark: input data size varies from 128 MB to 16 GB; the number of nodes is 16. Blue stacks are the reported performance for each MapReduce phase on Craysun. Red stacks are the reported performance for each MapReduce phase on YARNsim. Error is the accumulated error rate between Craysun and YARNsim.

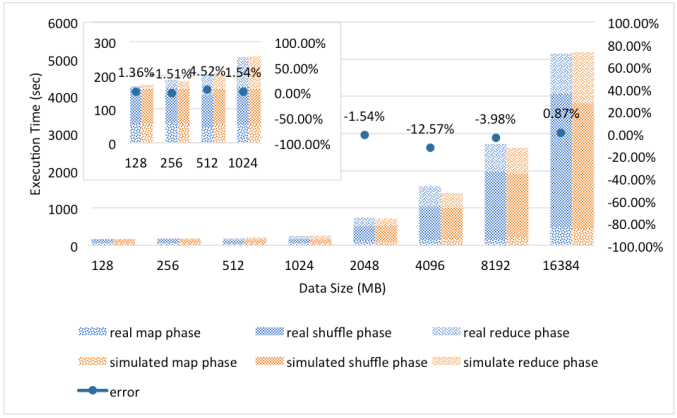

Figure 12. Performance comparison between Craysun and YARNsim on Wordcount benchmark: input data size varies from 128 MB to 16 GB; the number of nodes is 16. Blue stacks are the reported performance for each MapReduce phase on Craysun. Red stacks are the reported performance for each MapReduce phase on YARNsim. Error is the accumulated error rate between Craysun and YARNsim.

C. Hadoop Synthetic Benchmarks

We chose Terasort and Wordcount for our experiments because both benchmarks are widely accepted and can represent a class of Hadoop applications. To further analyze the application performance, we decomposed each job into three phases—map, shuffle, and reduce, assuming that the map phase and reduce phase contain the merge-sort operations. In Craysun and HEC, we used the data generated from the Teragen benchmark on 16 nodes and varied the input data size from 128 MB to 16 GB. To accurately record the performance of each phase in the real system, we leveraged the job history service provided by Hadoop, in which the detailed performance for each phase is reported. We collected these numbers and compared them with the numbers collected from the YARNsim system. In YARNsim, we used the same configuration as in Craysun and HEC for configuring the simulated clusters. We also built models for the Terasort and Wordcount benchmarks and ran them separately on the two different simulated clusters.

We compared the performance between the simulated jobs and real jobs for each phase and report the results in Figs. 11–14.

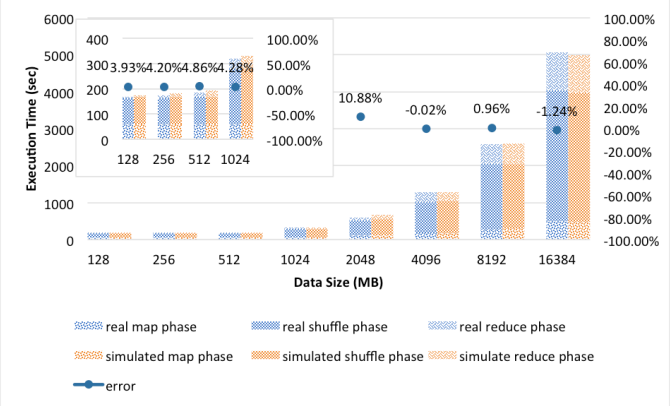

Figure 13. Performance comparison between HEC and YARNsim on Terasort benchmark: input data size v aries from 128 MB to 16 GB; the number of nodes is 16. Blue stacks are the reported performance for each MapReduce phase on HEC. Red stacks are the reported performance for each MapReduce phase on YARNsim. Error is the accumulated error rate between HEC and YARNsim.

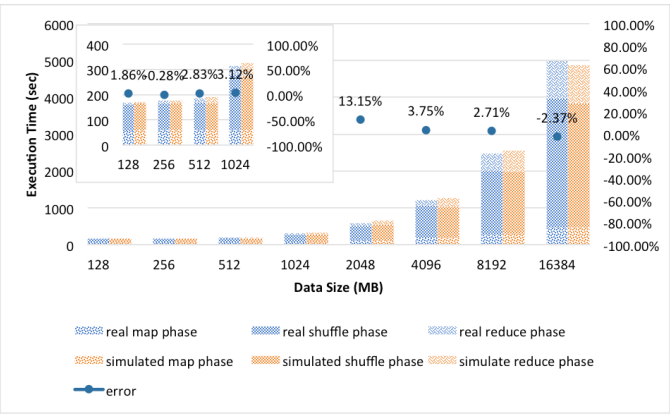

Figure 14. Performance comparison between HEC and YARNsim on Wordcount benchmark: input data size varies from 128 MB to 16 GB; the number of nodes is 16. Blue stacks are the reported performance for each MapReduce phase on HEC. Red stacks are the reported performance for each MapReduce phase on YARNsim. Error is the accumulated error rate between HEC and YARNsim.

As illustrated in Fig. 11, YARNsim can match the real system performance well in most cases, the maximum error rate between YARNsim and Craysun being 11.29%. As shown, the shuffle phase is the major part of overall performance. As we can see from the stacked bars in Fig. 11, YARNsim matches well in almost all test cases. The shuffle phase is dominated by network transfers and hence is a good representation of network accuracy. We show through this experiment that CODES-net is suitable for performance matching in YARNsim. We report the performance of Hadoop YARN on small data sets in the subfigure of Fig. 11. The purpose of these tests is to validate the Hadoop system overhead. As we can see, YARNsim captures these overheads accurately on the light-weight experiments. In Fig. 12, we use the Wordcount benchmark and report similarly to Fig. 11. The maximum error rate is 12.57%. We note that for some test cases, the reduce phase generates relatively large errors. We attribute these to a set of factors including the inaccurate slow.start.rate, skewed data, and computation in the reduce phase. To pinpoint the reason behind this phenomenon, we need to conduct further experiments. In this set of experiments, however, the reduce phase time constitutes only a small portion of the overall execution time. Thus the overall error rates are still acceptable for the majority of test cases.

In Figs. 13 and 14, we reports the experiments on HEC. Here, we use configurations similar to those used in the Craysun experiments. The experimental results are as expected, and the maximum error rate is 13.15%. The shuffle phase is still the overall peroformance bottleneck.

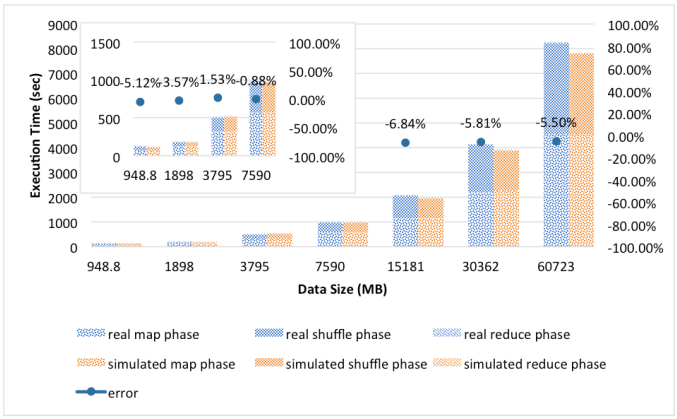

Figure 15. Performance comparison between HEC and YARNsim on bioinformatic application: input data size varies from 948.8 MB to 60.7 GB; the number of nodes is 16. Blue stacks are the reported performance for each MapReduce phase on HEC. Red stacks are the reported performance for each MapReduce phase on YARNsim. Error is the accumulated error rate between HEC and YARNsim.

D. Bioinformatics Application

In the field of bioinformatics, large dataset clustering is a challenging problem. Many biological scientists resort to Hadoop MapReduce for large-scale and parallel processing solutions. For example, researchers from the University of Delaware have developed an octree-based clustering algorithm for classifying protein-ligand binding geometries [32][33]. The proposed method is implemented in Hadoop MapReduce and is divided into a two-phase MapReduce workflow. The geometry reduction and key generation constitute the first-phase MapReduce job, where large datasets are read by the map tasks. The output of the first phase is the input of the second-phase MapReduce job. Here, an iterative octree-based clustering algorithm is implemented as a chain of MapReduce jobs indicating that the search has iterated to the deep level of the search tree. In the first phase, the output data size is about 1% of the input data size. Thus the MapReduce job spends most of its time on the map and shuffle phases.

We used this application as an exemplar for YARNsim. To model the application, we identified the sizes and locations of all data blocks in each phase and used them as input to the modeled MapReduce jobs. We varied the input file of protein geometry data from 948 MB to 60 GB and ran the experiments on HEC using 16 nodes. We also built a model for this clustering application and ran it on YARNsim with different configuration. The performance on HEC and YARNsim are

reported in Fig. 15. As illustrated, models based on YARNsim can help capture the application performance and thus provide the potential to further optimize the application and its environment. The clustering application used in this paper has distinct features. For instance, its map phase is significantly longer than the reduce phase because the data size in the reduce phase is only 1% of the input data size in the map phase. YARNsim faithfully captures this feature and can predict the application performance under different input data size. The maximum error is 6.84%.

V. Related Work

A plethora of community efforts on Hadoop system simulations have been conducted in the past few years, including Mumak [4], MRsim [8], SimMR [5], Hsim [33], MRSG [7], SimMapReduce [34], SLS [11], Starfish [9], and MRperf [6]. Most of the existing simulation systems focus on the first-generation Hadoop system. Since Hadoop YARN is relatively new to the community, few efforts have focused on the simulation of YARN. The SLS simulator targets Hadoop YARN; however, it focuses only on simulating job and task scheduling. Compared with SLS, YARNsim provides a more comprehensive view of the system; it includes the scheduling component and a detailed model for network and disk. First-generation Hadoop system simulators cannot handle the performance modeling for Hadoop YARN because of the fundamental changes in resource management and job/task management: in particular, the resource manager (ResourceManager) and application master (MRAppMaster), have replaced the JobTracker. Therefore, the corresponding protocols used in Hadoop YARN are different from the first generation Hadoop.

The prior work can also be categorized from the perspective of simulation techniques. The majority of the simulation systems are based on discrete-event simulation; few works are based on analytical models such as the Starfish project. The convenience of using discrete-event simulation is obvious. Hadoop system internal design is based on an event-driven model and state machine model. It is natural to use discrete events to model the flow logic flow and the change of states of Hadoop system components. Mumak, MRsim, SimMR, MRSG, and MRperf belong to this category. For modeling large-scale systems, the analytical method can be fast and efficient; however, it is constrained in terms of modeling complex scenarios. For example, the I/O system is nonlinear in terms of I/O request turnaround time. It is therefore unwarranted to use a constant bandwidth to get the I/O request response time. For systems based on discrete-event simulation, the simulation engine is sequential, meaning there is only one event queue for processing the model. Therefore, the simulation performance is largely constrained in modeling large-scale scenarios. YARNsim is based on parallel simulation and has the potential to simulate large-scale systems in a tractable amount of time.

In terms of simulating Hadoop system components, many simulation systems focus only on the MapReduce engine and lack a detailed model for HDFS. MRperf, MRSG, SimMR, and MRsim fall into this category. Mumak and SLS focus on the job and task-scheduling component and thus care less about the networking and disk models. Compared with these simulation systems, YARNsim is a comprehensive simulation system for Hadoop YARN. Its modules include both the software stacks and hardware stacks.

MRperf [6] is a discrete-event simulation system based on ns2 simulation framework. Compared with simulators such as Mumak, SimMR, and MRSim, MRPerf is both application aware and resource contention aware because of the detailed networking model. However, MRperf lacks a detailed model for hardware and HDFS. Moreover, because ns2 is based on sequential discrete event simulation, the performance of MRperf is determined by ns2. It is thus impractical to simulate large scale system using MRperf.

Starfish [9] is based on analytical performance models of the MapReduce system. It provides a what-if analysis mechanism for analyzing the complex parameter spaces in the Hadoop system. The system includes models for CPU, memory, network and disk. It also considers a rich set of parameters from the Hadoop system. The user is able to modify the parameters and observe the impact on system and application performance. However, the network and disk models used are simplified as the divisions and multiplications of a set of parameters. This kind of abstraction makes the simulation fast and efficient but loses the accuracy in the local model.

VI. Conclusions

In this paper, we present YARNsim, a Hadoop next-generation simulation system. YARNsim consists of four core modules: an application module, a YARN module, a MapReduce module, and an HDFS module. YARNsim also includes a set of hardware models. Different modules are functionally independent but operationally connected; together they capture different software and hardware layers in YARN and can replay YARN system behaviors with satisfactory granularity. YARNsim provides its users a comprehensive simulation platform where system architects can evaluate various design points and application developers can test/tune the application performance.

We validate the performance of YARNsim through a set of comprehensive Hadoop benchmark tests, including Terasort, Teragen, and Wordcount, on both simulation systems and the real-world clusters. We further validate YARNsim through a bioinformatics application. The experiment results show less than 10% error for most of the test cases.

YARNsim is not without its limitations. The current version of YARNsim does not support fault tolerance models. Thus, the user will not be able to simulate job execution under failure. Additionally, the HDFS module cannot yet model the secondary NameNode. We plan to incorporate a mechanism to support system components failure model and simulation. YARNsim also includes only the FIFO scheduling algorithm. We plan to develop a capacity scheduler, fair scheduler, and other advanced scheduling algorithms in the near future to support the simulation of complex job and task execution.

AKNOWLEDGMENT

We thank Boyu Zhang and Dr. Michela Taufer from University of Delaware for providing the MapReduce application and test data. This material was based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

REFERENCES

[1] "Apache Hadoop," *http://hadoop.apache.org/*. .

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[4] "Mumak: Map-Reduce Simulator," *Mumak: Map-Reduce Simulator*. .

[5] A. Verma, L. Cherkasova, and R. H. Campbell, "Play It Again, SimMR!," in *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, 2011, pp. 253–261.

[6] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in MapReduce setups," in *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09*, 2009, pp. 1–11.

[7] W. Kolberg, P. de B. Marcos, J. C. S. Anjos, A. K. S. Miyazaki, C. R. Geyer, and L. B. Arantes, "MRSG – A MapReduce simulator over SimGrid," *Parallel Comput.*, vol. 39, no. 4–5, pp. 233–244, Apr. 2013.

[8] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "MRSim: A discrete event based MapReduce simulator," in *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2010, vol. 6, pp. 2993–2997.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics," in *In CIDR*, 2011, pp. 261–272.

[10] H. Herodotou, F. Dong, and S. Babu, *MapReduce Programming and Cost-based Optimization? Crossing this Chasm with Starfish*. .

[11] "Yarn Scheduler Load Simulator (SLS)," *Yarn Scheduler Load Simulator (SLS)*. .

[12] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Commun ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[13] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[14] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, New York, NY, USA, 2013, pp. 5:1–5:16.

[15] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller, "MRONLINE: MapReduce Online Performance Tuning," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, New York, NY, USA, 2014, pp. 165–176.

[16] S. He, X.-H. Sun, and B. Feng, "S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems," in *2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS)*, 2014, pp. 514–523.

[17] S. He, X.-H. Sun, B. Feng, X. Huang, and K. Feng, "A cost-aware region-level data placement scheme for hybrid parallel I/O systems," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–8.

[18] S. A. McKee, "Reflections on the Memory Wall," in *Proceedings of the 1st Conference on Computing Frontiers*, New York, NY, USA, 2004, p. 162–.

[19] "Presto: Interacting with petabytes of data at Facebook," *https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920*. .

[20] "Gartner Report," *http://www.gartner.com/newsroom/id/2313915*. .

[21] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proceedings of the ACM SIGCOMM 2011 Conference*, New York, NY, USA, 2011, pp. 98–109.

[22] K. Shvachko, "HDFS Scalability: The Limits to Growth," Apr-2010.

[23] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: a high-performance, low memory, modular time warp system," in *Fourteenth Workshop on Parallel and Distributed Simulation, 2000. PADS 2000. Proceedings*, 2000, pp. 53–60.

[24] "CODES: Enabling Co-Design of Multilayer Exascale Storage Architectures." [Online]. Available: http://www.mcs.anl.gov/project/codes-enabling-co-design-multilayer-exascale-storage-architectures. [Accessed: 24-Oct-2014].

[25] N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross, "Model and simulation of exascale communication networks," *J. Simul.*, vol. 6, no. 4, pp. 227–236, Nov. 2012.

[26] N. Liu and C. D. Carothers, "Modeling Billion-Node Torus Networks Using Massively Parallel Discrete-Event Simulation," in *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, Washington, DC, USA, 2011, pp. 1–8.

[27] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *In Proceedings of the 2012 IEEE Conference on Massive Data Storage*, 2012.

[28] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, "Modeling a Million-Node Dragonfly Network Using Massively Parallel Discrete-Event Simulation," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 366–376.

[29] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model&Mdash;One Step Closer Towards a Realistic Model for Parallel Computation," in *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, New York, NY, USA, 1995, pp. 95–105.

[30] W. Tang, J. Jenkins, F. Meyer, R. Ross, R. Kettimuthu, L. Winkler, X. Yang, T. Lehman, and N. Desai, "Data-Aware Resource Scheduling for Multicloud Workflows: A Fine-Grained Simulation Approach," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014, pp. 887–892.

[31] D. Olker, *Optimizing NFS Performance: Tuning and Troubleshooting NFS on HP-UX Systems*. Prentice Hall Professional, 2002.

[32] T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, and M. Taufer, "A scalable and accurate method for classifying protein–ligand binding geometries using a MapReduce approach," *Comput. Biol. Med.*, vol. 42, no. 7, pp. 758–771, Jul. 2012.

[33] B. Zhang, D. T. Yehdego, K. L. Johnson, M.-Y. Leung, and M. Taufer, "Enhancement of accuracy and efficiency for RNA secondary structure prediction by sequence segmentation and MapReduce," *BMC Struct. Biol.*, vol. 13, no. Suppl 1, p. S3, Nov. 2013.

[34] Y. Liu, M. Li, N. K. Alham, and S. Hammoud, "HSim: A MapReduce Simulator in Enabling Cloud Computing," *Future Gener Comput Syst*, vol. 29, no. 1, pp. 300–308, Jan. 2013.

[35] F. Teng, L. Yu, and F. Magoulès, "SimMapReduce: A Simulator for Modeling MapReduce Framework," in *2011 5th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE)*, 2011, pp. 277–282.