Exploring Performance Models of Hadoop Applications on Cloud Architecture

Xing Wu
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec, Canada
w_xin21@encs.concordia.ca

Yan Liu
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec, Canada
yan.liu@concordia.ca

Ian Gorton
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
igorton@sei.cmu.edu

ABSTRACT
Hadoop is an open source implementation of the MapReduce programming model, and provides the runtime infrastructure for map and reduce functions programmed in individual applications. Commercial clouds such as Amazon Elastic MapReduce provides the Hadoop architecture with IaaS support. In this architecture, the map and reduce functions are major determinants of end-to-end application latency, along with the framework components responsible for data access and exchange. In this paper, we aim to explore modeling methods that capture the performance characteristic and the semantics of a Hadoop architecture. We present our early results for modeling the performance of a Hadoop application given the design of map and reduce functions using Layered Queueing Network (LQN). We build two different LQN models to represent the data parallel computing of these functions and calibrate both models using monitored performance data. The output of both models produces converging results that are within ~10% of observed performance. From our modeling experience, we further discuss the issues of modeling Hadoop architecture using LQN in general and describe our future work.

Categories and Subject Descriptors
C.4 [**Computer Systems Organization**]: Performance of Systems – *modeling techniques;* D.4.8 [**Software**]: Performance – *modeling and prediction, queueing theory.*

General Terms
Measurement, Performance, Design

Keywords
Performance modeling; Layered queueing network, MapReduce

1. Introduction
The demands of contemporary data analysis tasks continue to grow, both in scale and complexity, at unprecedented rates. High fidelity sensors, instruments and Internet applications are sources of incredibly rich data that can provide significant competitive advantages to Government and commercial organizations that are able to build high speed and volume data analysis applications. Insights gained from statistical models of massive data collections are already showing immense benefits in health care, insurance, and science, and the potential for is vast in areas such as military operations, intelligence analysis and logistics.

Data analysis at scale, or big data analysis, requires considerable compute resources to provide timely, useful answers. For this reason, the simplicity of the 'divide and conquer' MapReduce programming model [10], in particular its open source implementation Hadoop, has become the dominant approach used across all application domains. Hadoop provides abstractions for data-parallel programming, making it simple to build applications that analyze massive data collections by partitioning the data across clusters and enabling potentially thousands of tasks to process it concurrently.

Hadoop applications partition large data sets stored in the distributed Hadoop File System (HDFS) across a number of *mapper* tasks. Each mapper performs identical processing on its partition of the data, and produces an intermediate collection of key-value pairs. This intermediate data is then *shuffled* to *reducer* tasks, which aggregate the intermediate data to generate the final output. Application programmers are only required to write the map and reduce functions, with the Hadoop framework implementing the necessary mechanisms to reliably read, process and shuffle the data at scale. For more information on Hadoop, interested readers are referred to [11].

In general, the map and reduce functions are major determinants of end-to-end application latency, along with the framework components responsible for data access and exchange. Several performance studies also show that Hadoop has many sensitivity points that can affect performance (e.g. [30]). In fact, Hadoop offers approximately 200 configuration parameters that a programmer can adjust to improve an application's performance. Empirical evidence such as that in [32] shows how adjusting just a small number of these parameters can affect performance by 100-200%, often in non-intuitive ways due to non-linear interactions between parameter values. In total, this set of configuration parameters creates a complex multi-dimensional design space for programmers to navigate in order to reduce the latency of their codes. Evolution of the Hadoop framework is leading to elimination of some of these parameters [31], the majority of the parameters are required as the framework cannot make optimal decisions about their values in the absence of knowledge about the work an application will perform. Optimizing Hadoop applications on a given execution platform therefore remains a complex design task.

In this paper, we aim to explore a modeling method that captures the performance characteristic and the semantics of a Hadoop

QoSA'15, May 4–8, 2015, Montréal, QC, Canada.
Copyright © 2015 ACM 978-1-4503-3470-9/15/05…$15.00.
http://dx.doi.org/10.1145/2737182.2737197

architecture deployed in a Cloud environment. We focus on the following research questions

1) *What are the main entities that mostly contribute to performance characters of a Hadoop architecture?*
2) *Which modeling method should be used?*
3) *How to valid the soundness of a model in terms of capturing the semantics of MapReduce functions?*

To explore these research questions, we present two approaches using Layered Queueing Networks (LQN) models to represent the data parallel computing of map and reduce functions. Each model captures specific Hadoop configurations for runtime optimization such as shuffling intermediate data from map tasks to reduce data exchange latencies. We calibrate the models using monitored performance data collected from Amazon Web services including Elastic MapReduce (EMR) and EC2. We compare two models with the experiment results. The output of both models produces converging results that are within ~10% of observed performance. From our initial modeling experience, we further discuss the issues of modeling Hadoop architectures in general.

The paper is structured as follows. Section 2 provides a review of existing work on analyzing MapReduce performance including system approaches, modeling and simulation. The research questions are addressed in Section 3 and 4. Section 3 introduces the Hadoop architecture in the Amazon cloud and highlights the entities to be modeled. Section 4 presents the modeling method using LQN models. The models are demonstrated with a movie recommendation application. Section 5 concludes the paper.

2. Related Work

A variety of approaches have been proposed in the literature for Hadoop application performance analysis, prediction and tuning. We categorize these approaches as optimization, simulation and performance modeling, and then briefly describe each of these in the following sections.

Traditional database engines achieve high performance queries by applying a range of query optimization techniques. In SQL-based databases, this is feasible because queries are expressed in constrained language and the formats and relationships of the data are precisely defined in a relational schema that provides the necessary metadata for optimization. Such optimization approaches are problematic for Hadoop-based systems as the map and reduce functions can be written in arbitrary programming languages (e.g. Java, Python). This makes them opaque to the Hadoop framework. There is also no way to maintain statistics on the input data as its structure is not fixed and defined by a schema[1].

MANIMAL [12] is an attempt to overcome these difficulties through the analysis of compiled Hadoop programs to detect optimization opportunities. The code analyzer in MANIMAL looks for functionality in the map() that is equivalent to an SQL SELECT or PROJECT statement, and seeks data compression opportunities. It also generates an indexing program that builds a B+ Tree based index over the program's input data. The output from the optimizer is an execution descriptor that informs the MANIMAL execution fabric, a modified version of Hadoop, which applies optimizations and indexes during execution. Validation of the approach yield 3x to 11x speedups on a sample set of benchmarks. This shows the potential of the approach, but its ability to detect optimizations in arbitrary complex Hadoop programs has not been explored.

[1] Higher-level languages for building Hadoop applications such as HiveQL and Pig do enable schemas to be associated with input and intermediate data.

Hadoop++ [13] extends Hadoop to perform index accesses whenever a MapReduce job can exploit the use of indexes, and to co-partition data to enable maps to compute joins. The overall Hadoop framework is left unaltered and benchmarks show impressive speedups. However, programmer intervention is needed and hence the approach is not transparent to Hadoop developers. HadoopDB [14] also extends the basic framework to incorporate local relational databases, creating a parallel database system that uses Hadoop at its core. HadoopDB pushes SQL statements to the local databases and computes partial aggregates, which require a single reduce task for the final aggregation. HadoopDB therefore enforces a hybrid programming for applications, and is not useful for optimizing *vanilla* Hadoop codes.

Simulation is a proven technique for performance prediction and analyzing design trade-offs in a large number of software and hardware domains. It is not surprising therefore to see a number of efforts that have focused on building simulations of the Hadoop framework and applications. These include MRSim [19], HSim [18], Mumak [17], SimMR [16] and MRPerf [15].

MRSim is based on a composition of the discrete event simulation package SimJava [20] and GridSim [21], with the latter used to simulate network topology and traffic. The core abstractions in the simulator are models of CPU, HDD and Network Interface, which are replicated to describe a topology for cluster running Hadoop. Hadoop job descriptors specify the number of map and reduce tasks, data layout, algorithm description, and the job configuration parameters. The data layout describes the location of the data splits, including replicas, on the simulated nodes. Algorithm information is coarsely specified by measures including the number of CPU instructions per record and average record sizes of the data used in map and reduce tasks. Simple evaluations on a 4 node cluster for word count show promising results in terms of accuracy of predicting job characteristics, including latency. However, no extensive evaluation has been described for large scale cluster and complex analysis and data types.

HSim also aims to provide comprehensive Hadoop simulation capabilities. It is designed to enable a deep exploration of the design space for Hadoop jobs by supporting the modeling of an extensive number of parameters concerned with processing nodes, clusters, the Hadoop framework itself, and simulator controls that can govern simulation speeds and the fidelity of results. Cluster characteristics and Hadoop job behaviors are specified by populating a collection of parameters that are read by the simulator. Evaluation is performed firstly against a number of published industry benchmarks executed on clusters ranging from one to 100 nodes, and also against two Hadoop applications, namely information retrieval and content based image annotation. These two applications were implemented on a 4 node cluster, and comparisons against HSim predictions showed strong congruence as the number of mappers increased.

Mumak and SimMR focus on simulating the Hadoop task scheduling algorithms. Both simulate Hadoop workloads based on traces from Hadoop jobs. Mumak runs the actual Hadoop scheduler in a simulated environment, whereas SimMR simulates both the scheduler and the Hadoop jobs themselves. Due to their focus on scheduling tasks, there is no detailed simulation of the map and reduce phases in these simulators, and hence their ability to accurately simulate resource contention and the effects of various Hadoop configuration settings is limited.

MRPerf [15], based on the ns-2 discrete event simulator (http://en.wikipedia.org/wiki/Ns-2), also performs detailed simulations of Hadoop jobs. MRPerf aims to provide fine-grained simulation of Hadoop framework behavior. MRPerf requires detailed specifications of a cluster topology, application characteristics, and the layout of the application input and output data, all provided in XML. A major limitation is that it assumes simple map and reduce tasks, where the computing requirements are dependent on the size of the data, and not the processing of its content. MRPerf has been validated in several studies (TeraSort, Search and Index) using a 40 node cluster, and has been used to predict the effect of topology changes on a simulated 72 node cluster. Others however have reported difficulties in using MRPerf and have not been able to produce accurate performance predictions [19].

In general, the major problem with simulation as a technique for Hadoop performance prediction is accurately modeling the complexity of the map and reduce phases. This is typically done by specifying the amount of 'work' per unit of input data, where work is represented by CPU instructions, or some unit of time depending on the precise nature of the simulator. This information can only be coarsely estimated by an application designer in the absence of a running application. Simulators are also sensitive to changes in the implementation of the Hadoop framework. These changes can invalidate the simulations, necessitating changes to the core simulation approaches in order to accurately model framework behavior. This makes it difficult to keep simulators 'in sync' with Hadoop, and hence makes their usage of questionable value.

High fidelity performance models can potentially facilitate the prediction of Hadoop application performance, and provide faster solutions than simulation. For example, [33] takes a fine grained data-flow approach that provides a mathematical model of every stage of an Hadoop job. The models calculate the amount of data flowing through the different phases of a job, as well as the execution time for all tasks and phases. The model requires a large number of input parameters specifying both the job characteristics and cluster configuration. While [33] does not address how the parameters are obtained or model validation, [34] builds on this model and uses application profiling from a running application to populate the model. It then introduces a cost based optimizer that can predict the application performance given different job configuration parameters. Validation of the 'what-if' predictions is provided for a single application only, and this approach relies upon having the application code available for profiling and model calibration.

[35] presents a Hadoop job analyzer and prediction model, and uses a locally weighted regression method to train the prediction model based on historical traces. The analyzer measures data input size and the number of input records of a collection of sample applications, and tries to estimate the complexity of the map and reduce functions and the ratio of input values that are passed from the map to reduce phase. This data is then used for model training. Predicting an application's performance requires the application code to be profiled so that jobs in the training set with similar profiles can be used for prediction. A limited validation using *word count* shows that the method works well only when a job profile closely matches those in the training set. The paper does not describe in detail the experimental setup and conditions for prediction.

3. The Hadoop Architecture in Cloud

Apache Hadoop is an open-source implementation of the MapReduce programming model [10]. Amazon Web Services (AWS) provides the essential infrastructure components for setting up the Hadoop platform and running applications in a hosted cloud environment. AWS allows flexible choices of virtual machine images, capacities of virtual machine instances, and associated services (such as storage, monitoring and billing). The basic infrastructure components required to run Hadoop in AWS are listed in Table 1.

Table 1 Infrastructure Components

Platform	Apache Hadoop
Amazon Web Service	Simple Storage Service (S3)
	Elastic MapReduce (EMR)
Monitoring Tools	Amazon CloudWatch

The AWS deployment architecture is shown in Figure **1**. The architecture uses Elastic MapReduce (EMR), which is a Hadoop platform that comprises a resizable cluster of Amazon Elastic Compute Cloud (EC2) instances. In the EMR configuration, the number of core instances can be specified for running Hadoop, and a master instance manages the Hadoop cluster. The master assigns tasks and distributes data to core and task instances. Core instances run map/reduce tasks and store data using HDFS.

Instances can be chosen based on their desired capacity. For example, instances that are of the type of *m1.small* have 1.7 GB of memory, 1 virtual core of an EC2 Compute Unit, and 160 GB of local instance storage. A significant difference between Amazon EMR and a standalone Apache Hadoop is that with EMR both the input and output data are stored on Amazon S3 storage platform rather than an HDFS file system.

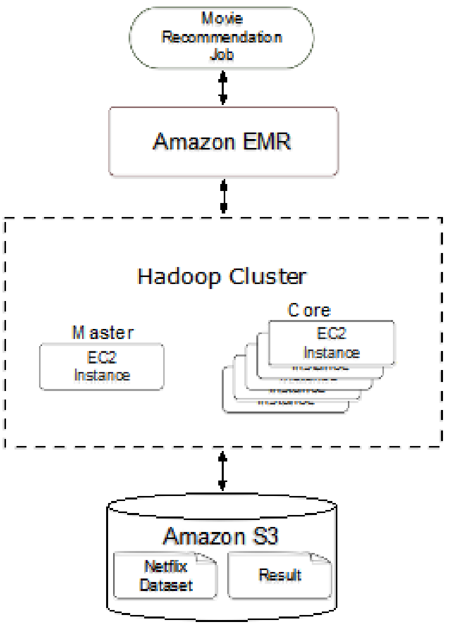


Figure **1** Deployment Architectures on AWS

As an illustrating example in Figure **1**, a movie recommendation application using Netflix datasets is launched as a Hadoop job on Amazon EMR. The MapReduce engine in Hadoop consists of one *JobTracker* and multiple *TaskTrackers*. When Hadoop jobs are submitted to the *JobTracker*, they are split into *map* tasks and *reduce* tasks. These tasks are then pushed to available *TaskTracker* nodes. Each *TaskTracker* has a number of map slots and reduce slots. Every active map or reduce task occupies one slot.

As the reduce tasks need to process the output data of map tasks, the data processing of reduce tasks cannot start until all the map tasks have finished. However, the shuffling of intermediate result data from map tasks to reduce tasks can start much earlier than the actual reduce processing. Hence reduce tasks occupy allocated reduce slots and begin data shuffling when some percentage of the map tasks are complete – this is known as a *slow start* of reduce tasks. The parameter *mapred.reduce.slowstart.completed.maps* specifies the percentage of map tasks that must be completed before the shuffling of data to reduce tasks can start.

4. The Modeling Method

4.1 Introduction to LQN Model

4.1.1 Queueing Networks

A queueing network (QN) consists of several interconnected queues. Figure 2 is an example of a QN with three nodes. Each node contains a queue and a processor. Nodes are connected by arrows that stand for requests and responses.

A QN model describes the flow of requests and predicts the waiting times. The method is based on the Little's theorem as shown in Equation (1).

$$N = \lambda T \qquad (1)$$

where N is the average length of the queue; λ is the average throughput; and T is the average service time for a request.

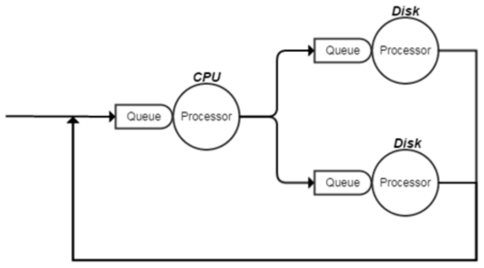


Figure 2 An example queueing network model

4.1.2 Layered Queueing Network

Layered Queueing Network (LQN) models extend QN models to support nested queueing networks, which are suitable for performance modeling of complex distributed systems [22]. In a LQN model, the service time of each request is measured by the response time of a queueing network. For example, when a HTTP request is sent to a web server, it waits in the request queue until a thread is available to process it. After the web server starts to process the request, the HTTP request waits for the amount of time it takes to be processed by the application server. Hence there are three layers of tasks as Figure 3 shows, namely the client, the HTTP server and the application server.

The notation of LQN models includes *Task*, *Processor*, *Entry*, *Call*, and *Activity*. In Hadoop, the software components that interact with others are modeled as *Tasks*. The software components that provide services are *Entries*. A *Task* can have multiple *Entries*. The *Entries* can *Call* (or send requests to) *Entries* of other *Tasks*. A *Call* can be synchronous, asynchronous

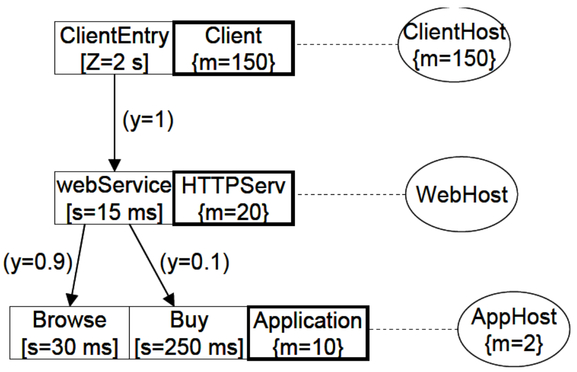


Figure 3 An example of layers from [22]

or forwarding. If a *Task* does not receive any *Calls* from other *Tasks*, it is defined as a *Reference Task*. When the operations of a *Task* become complex, the *Task* can be modeled as a precedence graph of *Activities*. An *Activity* can also call one or more *Entries* of other *Tasks*. The physical entities that perform the operations are *Processors*.

To model application workload, the following parameters need to be specified for *Activities* and *Entries*:

- **s** – the time demand on the corresponding processor.
- **Z** – wait time (also called think time). This is used to model the time delay that does not occupy the processor.
- **y** – synchronous call to another *Entry*.
- **z** – asynchronous (no wait for response) call to another *Entry*.
- **f** – can be set to 0 or 1. Value 1 means the number of the requests to other *Entries* is deterministic. Value 0 means the number follows a geometric distribution with the stated mean.

For more details of LQN, please refer to [22] [23] [24].

4.2 Modeling MapReduce Functions

The objective of our modeling is to capture the main factors that contribute to the end-to-end processing time of Hadoop jobs. Hadoop is able to run different jobs concurrently, and a variety of schedulers exist to better utilize the resources [25] [10]. Hence the processing time of one job can be influenced by other concurrent jobs. In our models, we isolate each job by assuming that there is only one job running on a Hadoop cluster at a time.

Our modeling method is based on the Hadoop architecture of Amazon EMR. Our model needs to capture the following aspects to be faithful in representing the architecture:

- Application user and job entry
- Map and reduce functions in parallel
- Data input and output operations
- Data transfers over the network

We encapsulate each function of running Hadoop job as a *Task*. Each *Task* has an *Entry*. The invocation from one function to another is modeled through the calling edge from the *Task*'s *Entry*. Each *Task* runs on a *Processor* that characterizes a system level entity, such as a CPU processor. We also breakdown the sub-functions of map and reduce functions into a set of individual *Activities*.

With LQN, there are two ways of modeling the concurrency of map and reduce functions. An essential element of modeling these functions in Hadoop is capturing the *slow start* mechanism for reduce tasks as discussed in the previous section. We have investigated two approaches in LQN for modeling this mechanism, as described in the following subsections.

4.2.1 Non-priority Model

Figure 4 shows the LQN model of a Hadoop application deployed on Amazon EMR. In this model, all the *Calls* are synchronous, and the numbers of *Calls* are deterministic. The model contains five processors, namely *Client*, *EC2CPU*, *EC2Disk*, *S3* and *Network*. All process messages in a FIFO manner.

- *Client*: User who submits the MapReduce job.
- *EC2CPU*: The CPUs of Amazon EC2 instances.
- *EC2Disk*: The local disks of Amazon EC2 instances.
- *S3*: The S3 storage service provided by Amazon.
- *Network*: All the network transportation.

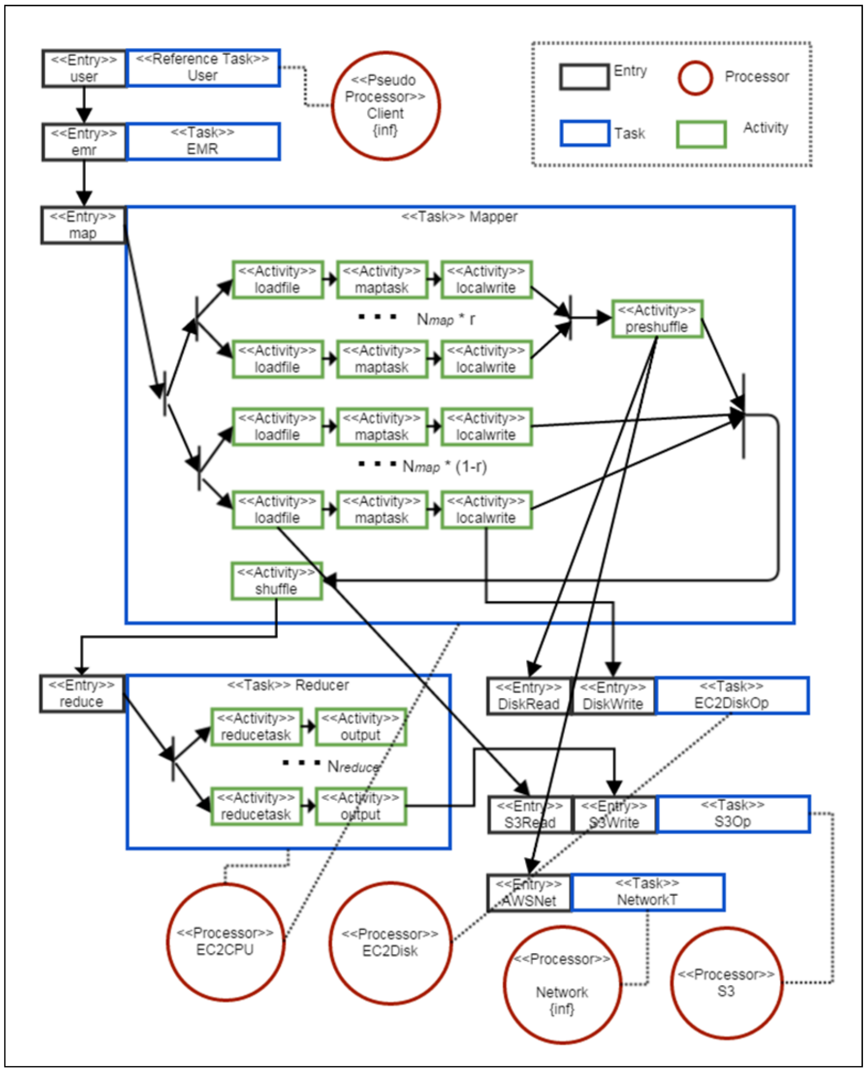


Figure 4 LQN Model of MapReduce Functions

The **application user** is indicated by the *Reference Task* at the top as *User*. This *Reference Task* models the EMR user who submits a Hadoop job. The **job entry** is modeled by the *EMR Task* that calls the *Mapper Task*, which in turn calls the *Reducer Task*. This means the *EMR Task* will be blocked until both the *Mapper Task* and *Reducer Task* end.

The ***Mapper*** *Task* contains the map and shuffle phases of a Hadoop job. The processor of the *Mapper Task* is the CPUs of the Amazon EC2 instances. Each *Mapper Task* is scheduled on a map slot that consists of three *Activities*:

1. *loadfile*: Load input file partition from Amazon S3. It calls the *S3Op Task* with the *S3Read* entry.
2. *maptask*: User-defined map processing on the input file.
3. *localwrite*: If the intermediate result from the map processing is bigger than a specified *spill* threshold, it will be written to the local disk of the map slot. It calls the *EC2DiskOp Task* with the *DiskWrite Entry*.

The concurrency of the *Mapper Task* is modeled using the *and-fork* notation of LQN. An *and-fork* specifies the start of a concurrent execution. The number of forked paths is the number of current executions of the map tasks in a Hadoop job.

To represent the *slow start* mechanism, we model two groups of map tasks. The number of tasks of each group is given as *Nmap*r* that depends on the *slow start* ratio *r* and the total amount of map tasks *Nmap*. For example, if *r* is 0.5, then the shuffling starts when half of the map tasks finish. This is modeled by means of an LQN *and-join* and the *preshuffle Activity* in the *Mapper Task* that represents the shuffle phase. When all the map tasks finish, the *shuffle Activity* begins. It calls the *DiskRead Entry* of the *EC2DiskOp Task* to read files and then calls the *AWSNet Entry* of the *NetworkT Task* to transfer data through network.

The ***Reducer*** *Task* models the reduce tasks running on the reduce slots of a Hadoop cluster. A *Reducer Task* consists of a *reducetask Activity* and an *output Activity*.

- *reducetask*: User-defined reduce processing.
- *output*: Output the result of reduce tasks to Amazon S3. It calls the *S3Write Entry* of the *S3Op Task*.

4.2.2 Priority Model

The *slow start* mechanism can be modeled using a specific feature of LQNs called *Priority*. The priorities of tasks determine their execution order by the processor. Tasks with priorities higher than the task currently running on the processor preempt the running task.

The *Mapper Task* of the non-priority model in Figure 4 is revised to add two *Tasks Map1 and Map2* in a lower layer*,* as shown in Figure 5. The other parts of the model remain the same as the non-priority model.

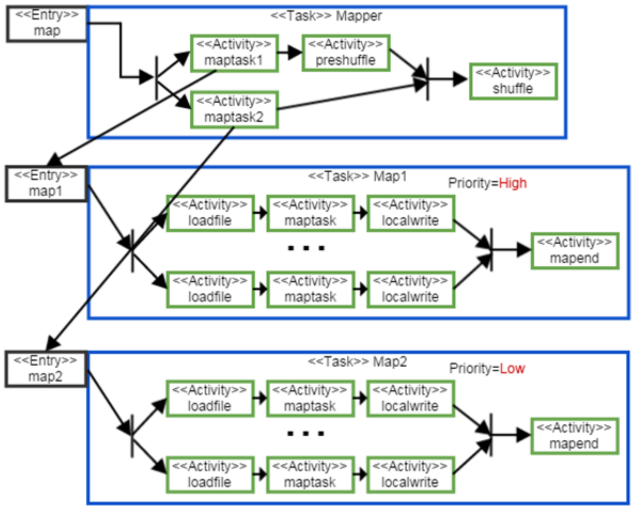


Figure 5 Priority Model of Mapper Tasks

Both *Map1 and Map2 Tasks* contain the same *Activities* as presented in the non-priority model. They run concurrently. The goal is to achieve the synchronization for *slow start* so that when *Map1 Task* ends, the shuffling of data begins and when both *Map1 and Map2 Tasks* complete, the reduce task can start. Hence we set the *Map1 Task* with higher priority than *Map2 Task*. As a result, we can represent the *slow start* feature by ensuring the *maptask1 Activity* of the *Mapper Task* in the upper layer can start and finish before the *maptask2 Activity*. Likewise, the number of *Map1 Tasks* is *Nmap*r.*

4.3 LQN Model Specification

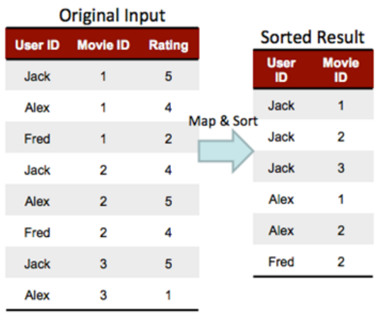
4.3.1 Movie Recommendation Application

We evaluate our models using a movie recommendation application we developed to process sample data sets from the Netflix Prize [26]. The application aims to provide a list of movies that most likely contain the movies a user may like. The sample data size is approximately 2GB in total. The data set contains 17,770 files. The MovieID ranges from 1 to 17,770 sequentially with one file per movie. Each file contains three fields, namely UserID, Rating and Date. The UserID ranges from 1 to 2649,429 with gaps. There are in total 480,189 users. Ratings are on a five star scale from 1 to 5. Dates have the format YYYY-MM-DD. We merged all the 17,770 files into one file, with each line in the format of "UserID MovieID Rating Date", ordered by the ascending date, to create the input data for our application.

We developed an item-based collaborative filtering algorithm [27] for the application. We define a *movie fan* as a user who rates a given movie higher than three stars. *Mutual fans* of two movies are the users who have rated both movies higher than three stars. Then we measure the similarity between two movies by the number of their mutual fans. As a result, we output a recommendation list that contains top-N most similar movies to each movie. Following this application logic, we present the implementation in Hadoop and highlight key artifacts in the MapReduce programming model.

In Hadoop, this application consists of three rounds of MapReduce processing as follows:

Round 1: Map-and-Sort the *user-movie* pairs. Each pair implies the user is a fan of the movie. A reducer is not needed in this round. Figure 6 shows an example of the input and output.


Original Input

User ID	Movie ID	Rating
Jack	1	5
Alex	1	4
Fred	1	2
Jack	2	4
Alex	2	5
Fred	2	4
Jack	3	5
Alex	3	1

Map & Sort

Sorted Result

User ID	Movie ID
Jack	1
Jack	2
Jack	3
Alex	1
Alex	2
Fred	2

Figure 6 An example for Round 1

Round 2: Calculate the number of mutual fans of each movie. Figure 7 demonstrates the processing with an example. Assume Jack is a fan of movie 1, 2, and 3, then movie 1 and 2 has one mutual fan as Jack. Likewise, movie 1 and 3, and movie 2 and 3 also have one mutual fan as Jack. A mapper finds all the mutual fans of each movie and outputs a line for every mutual fan. Then the reducer aggregates the result based on the movie pair that has one mutual fan and counts the number of mutual fans.

Round 3: Extract the movie pairs in the result of Round 2 and find the top-N movies that have the most mutual fans of each movie. The mapper extracts the movie IDs from the movie pairs and the reducer aggregates the result from the mapper and orders the recommended movies based on the numbers of their mutual fans. As Figure 8 demonstrates, movie pair 1-2 has mutual fan count of 2 and movie pair 1-3 has one mutual fan. Therefore the movie id 1 has mutual fans with movie id 2 and 3. The recommended movies for movie id 1 are movies [2,3] in descending order of the count of mutual fans.

Our deployment of this application uses AWS Elastic MapReduce (EMR). In the EMR configuration, we setup one

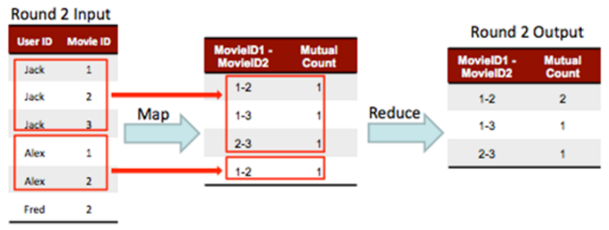


Figure 7 An Example of Round 2 processing

master instance and ten core instances to run the Hadoop implementation of the movie recommendation application. The master instance manages the Hadoop cluster. It assigns tasks and distributes data to core instances and task instances. Core instances run map/reduce tasks and access the movie input data from Amazon S3. All instances are in the type of m1.small, which has 1.7 GB of memory, 1 virtual core of EC2 Compute Unit, and 160 GB of local instance storage.

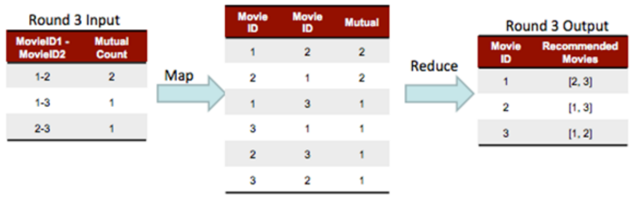


Figure 8 An Example of Round 3 processing

4.3.2 Model Calibration

We calibrate the non-priority model and the priority model based on the EMR configuration, monitored data from Amazon CloudWatch and the Hadoop log files produced. As the movie recommendation application contains three rounds of MapReduce jobs, we only take the data from Round 2 for the calibration. We choose Round 2 because it is the most computationally complex of the three rounds and takes the largest proportion of the application runtime.

4.3.2.1 Non-priority Model

The LQN model for the movie recommendation application contains five processors: *Client, EC2CPU, EC2Disk, S3* and *Network*. All use FIFO scheduling. We assume that there is only one Hadoop job running at a time on the cluster, so we configure the number of *Clients* to 1. As we have 10 core instances and each of them has only one CPU core, we set the multiplicity of *EC2CPU* to 10. The multiplicity of *Network* and *S3* are set to infinite.

We use the LQN solver developed at Carleton University [24]. The description of *Processors* is listed in Model Segment 1.

Model Segment 1 Processors

```
P 0
p EC2CPU f m 10
p Client f m 1
p EC2Disk f
p S3 f i
p Network f i
-1
```

For the *Tasks* in the model, all the *Tasks* are non-reference task except the *User Task*. The numbers of the *Mapper Task* and the *Reduce Task* are set based on the numbers of map slots and reduce slots. In this case, each instance has two map slots and one reduce slot. The model of *Tasks* is listed in Model Segment 2 .

Model Segment 2 Tasks

```
T 0
t User r user -1 Client z 33000000
t EMR n emr -1 EC2CPU
t Mapper n map -1 EC2CPU m 20
t Reducer n reduce -1 EC2CPU m 10
t EC2DiskOp n DiskRead DiskWrite -1 EC2Disk
t S3Op n S3Read S3Write -1 S3
t NetworkT n AWSNet -1 Network i
-1
```

In the Hadoop job of Round 2, the input file is 17*6 MB = 102 MB in total. Amazon EMR launches 51 map tasks and 17 reduce tasks. The default setting of the *slow start* ratio in EMR is 0.5. Thus the number of the first group of map tasks is *Nmap***r* = 51*0.5 = 26.

We calibrate the service time of the entries and activities based on the estimation of processing times from the Hadoop logs. For the service time of map tasks, we observe that the first 20 (we have 20 map slots) last 10 minutes while the 10 single-core EC2 instances are fully saturated. Therefore we can estimate the service time of a map task as 10 minutes/(20/10) = 5 minutes, which is 300,000 milliseconds as Model Segment 3 shows. The service time of reduce tasks are calculated in the same manner.

Model Segment 3 Service Time Settings

```
# service time setting
s DiskRead 50.0 -1
s DiskWrite 50.0 -1
s S3Read 50.0 -1
s S3Write 50.0 -1
s AWSNet 80.0 -1

s maptask1 300000
s reducetask12 180000
```

4.3.2.2 Priority Model

For the priority model, most calibrations are the same as the non-priority model. The main difference is that we change the scheduling strategy of the *EC2CPU* processor from FIFO to Priority. Model Segment 4 shows the modifications in the priority model of the processors.

Model Segment 4 Processors in Priority Model

```
P 0
p EC2CPU p m 10
p Client f m 1
p EC2Disk f
p S3 f i
p Network f i
-1
```

Then we set the priorities of each task that is processed by the *EC2CPU* processor as Model Segment 5 shows.

Model Segment 5 Tasks in Priority Model

```
T 0
t User r user -1 Client z 32700000
t EMR n emr -1 EC2CPU
t Mapper n map -1 EC2CPU 3
t Mapslot1 n map1 -1 EC2CPU 3 m 20
t Mapslot2 n map2 -1 EC2CPU 2 m 20
t Reducer n reduce -1 EC2CPU 1 m 10
t EC2DiskOp n DiskRead DiskWrite -1 EC2Disk
t S3Op n S3Read S3Write -1 S3
t NetworkT n AWSNet -1 Network i
-1
```

5. Results and Discussion

Our model[2] is solved by running the LQN simulator [24]. We ran each simulation twenty times to account for variation in the results. After tuning the performance of the Hadoop implementation on EC2, we ran the job three times, again to account for variability in performance, and averaged the results. Table 2 shows the average job execution times for both the model and the Hadoop application, along with the standard deviations calculated from all the runs. The average processing time includes all the three phases (map, shuffle, reduce) of Round 2 in the movie recommendation application. Given the standard deviation, both models estimate the processing time with similar accuracy.

Table 2 Performance Result of One-day Data

One-day Data	Amazon EMR	Non-priority Model	Priority Model
Average Processing Time (minutes)	40.3	44.5	40.7
STDEV (minutes)	4.9	2.9	3.1

The above model estimates the average processing time of the movie recommendation using one day's data from Netflix. We further calibratd the model to estimate the processing time of the movie recommendation using 3 days' data. As the input data size of 3 days is approximately 3 times of 1 day's data input, we make the assumption that the service time of map and reduce tasks in processing 3 day's data is approximately in the ratio of 3 times. We therefore modify the settings in Model Segment 3 as below. The simulation produces the result in Table 3 compared to the measurement in the AWS EMR environment. Likewise, both models produce reasonable estimations of the application execution time, with the simulation results having standard deviation of up to 9%. The error of estimation between modeled results and measurement is also approximately 9%. These results gives us more confidence that the models provide a useful estimation of end-to-end processing time.

```
# service time setting
s DiskRead 50.0 -1
s DiskWrite 50.0 -1
s S3Read 50.0 -1
s S3Write 50.0 -1
s AWSNet 80.0 -1

s maptask1 900000

s reducetask12 540000
```

Table 3 Performance Result of Three-day Data

Three-days Data	Amazon EMR	Non-priority Model	Priority Model
Average Processing Time (minutes)	153	138	136
STDEV (minutes)	Only run once due to budget constraints	12	9

Our calibration of the service time of map and reduce tasks are based on the entire time that the tasks last, and do not break down the processing to capture the framework overheads for spilling, merging and shuffling data. As many of these operations can happen concurrently, our model does not capture their overlap, which may explain the approximately 10% overestimation of both models compared to the actual measurement. We also need to investigate further the predictive capabilities of the two models to assess which gives better predictions for a range of applications. In this experiment both the non-priority and the priority model are calibrated with the same parameter values, and for this application there is no statistically significant difference between their predictions.

A further improvement would be to estimate the service times based on a representative sample Hadoop job performance characteristics, instead of calibrating parameters for a specific MapReduce job. The aim would be to estimate model parameters from the execution characteristics of a set of benchmark Hadoop applications, in a manner similar to [35]. Our successful performance prediction method from previous work provides a guide for the approach [28]. It remains our future work to apply this method to Hadoop applications.

In addition to the model calibration, other challenges remain in the performance modeling, such as how to model the data skew phenomena in LQN. Data skew is a typical problem in big data analysis [29]. When data skew occurs, some tasks take much longer than others and eventually impact the whole processing time of a MapReduce job. The cause of data skew relates to the data characteristics as well as the configuration of the Hadoop framework. These factors have not been explored in our current models.

6. Conclusion

In this paper, we propose two LQN models to capture the concurrent executions of map and reduce functions. We calibrate the two models with monitored performance data from a movie recommendation application running on Amazon EMR. Both models provide reasonable predictions on the processing time, which shows our performance modeling method encapsulates the essential factors that contribute to overall Hadoop application performance. We demonstrate the modeling method on a commodity deployment architecture using a commercial Hadoop platform, hence we believe this method is applicable to other Hadoop applications. To improve the accuracy of predictions, we plan to investigate new approaches for calibrating service times based on sample data from a collection of Hadoop benchmark applications that would characterize the performance of a given target execution cluster.

7. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.

[2] Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= http://doi.acm.org/10.1145/332040.332491.

[4] Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[2] The source files of our models are available for downloading at http://xingwu.me/downloads/HadoopLQN.zip.

[5] Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.

[8] Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= http://dx.doi.org/10.1016/j.jss.2005.05.030.

[9] Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= http://doi.acm.org/10.1145/90417.90738.

[10] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, January 2008.

[11] http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[12] Eaman Jahani, Michael J. Cafarella, and Christopher Ré. 2011. Automatic optimization for MapReduce programs. Proc. VLDB Endow. 4, 6 (March 2011), 385-396.

[13] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. 2010. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). Proc. VLDB Endow. 3, 1-2 (September 2010), 515-529.

[14] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. 2009. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc. VLDB Endow. 2, 1 (August 2009), 922-933

[15] Wang, Guanying, Ali Raza Butt, Prashant Pandey, and Karan Gupta. "A simulation approach to evaluating design decisions in mapreduce setups." In*Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pp. 1-11. IEEE, 2009.

[16] A. Verma, L. Cherkasova, and R. H. Campbell. Play It Again, SimMR! In 2011 IEEE International Conference on Cluster Computing, pages 253–261. IEEE, Sept. 2011.

[17] A. C. Murthy. Mumak: Map-Reduce Simulator. MAPREDUCE-728, Apache JIRA, Also available at http://issues.apache.org/jira/browse/MAPREDUCE-728, 2009.

[18] Yang Liu, Maozhen Li, Nasullah Khalid Alham, Suhel Hammoud, HSim: A MapReduce simulator in enabling Cloud Computing, Future Generation Computer Systems, Volume 29, Issue 1, January 2013, Pages 300-308, ISSN 0167-739X,

[19] Hammoud, S.; Maozhen Li; Yang Liu; Alham, N.K.; Zelong Liu, "MRSim: A discrete event based MapReduce simulator," Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on , vol.6, no., pp.2993,2997, 10-12 Aug. 2010

[20] http://www.dcs.ed.ac.uk/home/hase/simjava/

[21] Buyya, Rajkumar, and Manzur Murshed. "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing." Concurrency and computation: practice and experience 14.13-15 (2002): 1175-1220.

[22] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. "Enhanced modeling and solution of layered queueing networks."*Software Engineering, IEEE Transactions on* 35, no. 2 (2009): 148-161.

[23] Murray Woodside, Tutorial Introduction to Layered Modeling of Software Performance, 2013. http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialh.pdf

[24] Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, Alex Hubbard and Martin Mroz, Layered Queueing Network Solver and Simulator User Manual, 2013. http://www.sce.carleton.ca/rads/lqns/LQNSUserMan-jan13.pdf

[25] Zaharia, Matei, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. "Improving MapReduce Performance in Heterogeneous Environments." In *OSDI*, vol. 8, no. 4, p. 7. 2008.

[26] Netflix Prize. http://www.netflixprize.com/

[27] Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." Internet Computing, IEEE 7.1 (2003): 76-80.

[28] Yan Liu, Ian Gorton, Anna Liu, Ning Jiang, and Shiping Chen. 2002. Designing a test suite for empirically-based middleware performance prediction. In Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications (CRPIT '02). Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 123-130.

[29] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. 2012. SkewTune: mitigating skew in mapreduce applications. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 25-36.

[30] Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin, MapReduce and Parallel DBMSs: Friends or Foes?, CACM (2010).

[31] https://issues.apache.org/jira/browse/MAPREDUCE-64

[32] Shivnath Babu. 2010. Towards automatic optimization of MapReduce programs. In Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10). ACM, New York, NY, USA, 137-142.

[33] Herodotou, Herodotos. "Hadoop performance models." arXiv preprint arXiv:1106.0940, Technical Report CS-2011-05, Duke University (2011).

[34] Herodotou, Herodotos and Babu, Shivnath. "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs…" PVLDB 4, no. 11 (2011): 1111-1122.

[35] Song, Ge, Zide Meng, Fabrice Huet, Frederic Magoules, Lei Yu, and Xuelian Lin. "A Hadoop MapReduce Performance Prediction Method." In *2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. 2013