Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

An improved task assignment scheme for Hadoop running in the clouds

Wei Dai* and Mostafa Bassiouni

Abstract

Nowadays, data-intensive problems are so prevalent that numerous organizations in various industries have to face them in their business operation. It is often crucial for enterprises to have the capability of analyzing large volumes of data in an effective and timely manner. MapReduce and its open-source implementation Hadoop dramatically simplified the development of parallel data-intensive computing applications for ordinary users, and the combination of Hadoop and cloud computing made large-scale parallel data-intensive computing much more accessible to all potential users than ever before. Although Hadoop has become the most popular data management framework for parallel data-intensive computing in the clouds, the Hadoop scheduler is not a perfect match for the cloud environments. In this paper, we discuss the issues with the Hadoop task assignment scheme, and present an improved scheme for heterogeneous computing environments, such as the public clouds. The proposed scheme is based on an optimal minimum makespan algorithm. It projects and compares the completion times of all task slots' next data block, and explicitly strives to shorten the completion time of the map phase of MapReduce jobs. We conducted extensive simulation to evaluate the performance of the proposed scheme compared with the Hadoop scheme in two types of heterogeneous computing environments that are typical on the public cloud platforms. The simulation results showed that the proposed scheme could remarkably reduce the map phase completion time, and it could reduce the amount of remote processing employed to a more significant extent which makes the data processing less vulnerable to both network congestion and disk contention.

Keywords: Cloud computing; Hadoop; MapReduce; Task assignment; Data-intensive computing; Parallel and distributed computing

Introduction

We have entered the era of Big Data. It was estimated that the total volume of digital data produced worldwide in 2011 was already around 1.8 zettabytes (one zettabyte equal to one billion terabytes) compared to 0.18 zettabytes in 2006 [1]. Data has been generating in an explosive way. Back in 2009, Facebook already hosted 2.5 petabytes of user data growing at about 15 terabytes per day. And the trading system in the New York Stock Exchange generates around one terabyte of data every day. For many organizations, petabyte datasets have already become the norm, and the capability of data-intensive computing is a necessity instead of a luxury. Data-intensive computing lies in the core of a wide range of applications used across various industries, such as web indexing, data mining, scientific simulations, bioinformatics research, text/image processing, and business intelligence. In addition to large volume, Big Data also features high complexity, which makes the processing of data sets even more challenging. As a result, it is difficult to work with Big Data using most relational database management systems. And the solution is parallel and distributed processing on large number of machines.

MapReduce [2] is a parallel and distributed programming model and also an associated implementation for processing huge volumes of data on a large cluster of commodity machines. Since it was proposed by Google in 2004, MapReduce has become the most popular technology that makes data-intensive computing possible for ordinary users, especially those that don't have any prior experience with parallel and distributed data processing. While Google owns its proprietary implementation of MapReduce, an open source implementation named Hadoop [3] has gained great popularity in the rest of the world. Hadoop is now being used at

* Correspondence: wdai@knights.ucf.edu
School of Electrical Engineering & Computer Science, University of Central Florida, 4000 Central Florida Blvd., Orlando, Florida 32816, USA

many organizations in various industries, including Amazon, Adobe, Facebook, IBM, Powerset/Microsoft, Twitter, and Yahoo! [4]. Many well-known IT companies have been either offering commercial Hadoop-related products or providing support for Hadoop, including Cloudera, IBM, Yahoo!, Google, Oracle, and Dell [5].

The access to computer clusters of sufficient size is necessary for the parallel processing of large volumes of data. However, not every organization with data-intensive computing needs can afford or has the interest to purchase and maintain such computer clusters. The innovative concept of utility computing proposed a perfect solution to this problem, which eliminates both upfront hardware investment and periodical maintenance costs for cloud users. The combination of Hadoop and cloud computing has become an attractive and promising solution to parallel processing of terabytes and even petabytes datasets. A well-known feat of running Hadoop in clouds for data-intensive computing was the New York Times used 100 Virtual Machines (VM's) on Amazon Elastic Compute Cloud (EC2) [6] to convert 4 terabytes of scanned archives from the paper to 11 million articles in PDF format in less than 24 hours [7].

Although Hadoop has become the most prevalent data management framework for the processing of large volumes of data in clouds, there exist issues with the Hadoop scheduler that can seriously degrade the performance of Hadoop running in clouds. In this paper, we discuss the issues with the Hadoop task assignment scheme, and present an improved scheme, which is based on an optimal algorithm for minimum makespan scheduling and explicitly strives to shorten the duration of the map phase of MapReduce jobs. We conducted extensive simulations to evaluate the performance of the proposed scheme. The simulation results indicated that the proposed scheme could significantly improve the performance of Hadoop in terms of both the completion time of map phase and the amount of remote processing employed.

The rest of the paper is organized as follows. Background: MapReduce and Hadoop introduces the related background in MapReduce and Hadoop. Issues with the Hadoop task assignment scheme discusses issues with the Hadoop task assignment scheme in the context of the cloud environments. Related mathematical model introduces the related mathematical model on which our new scheme is based. The ECT task assignment scheme provides the details of the new scheme. Evaluation presents the simulation setup and results. Related work is introduced in Related work, and we conclude in Conclusion.

Background: MapReduce and Hadoop

In the programming model of MapReduce [2], the input of the computation is a set of key/value pairs, and the output is also a set of key/value pairs usually in a different domain from the input. Users define a map function which converts one input key/value pair to an arbitrary number of intermediate key/value pairs, and a reduce function which merges all intermediate values of the same intermediate key into a smaller set of values, typically one value for each intermediate key. An example of the application of the programming model is counting the number of occurrences of each word in a large collection of documents. The input < key/value > pair to the map function is < the name of certain document in the collection/contents of that document>. The map function emits an intermediate key/value pair of < word/1 > for each word in the document. Then the reduce function sums all counts emitted for a particular word to obtain the total number of occurrences of that word.

Hadoop [3] is currently the most mature, accessible, and popular implementation of the MapReduce programming model. A Hadoop cluster adopts the master–slave architecture, where the master node is called the JobTracker, and the multiple slave nodes TaskTrackers. Hadoop is usually supported by the Hadoop Distributed File System (HDFS), an open-source implementation of the Google File System (GFS). HDFS also adopts the master–slave architecture, where the NameNode (master) maintains the file namespace and directs client applications to the DataNodes (slaves) that actually store the data blocks. HDFS stores separate copies (three copies by default) of each data block for both fault tolerance and performance improvement. In a large Hadoop cluster, each slave node serves as both the TaskTracker and the DataNode, and there would usually be two dedicated master nodes serving as the JobTracker and the NameNode respectively. In the case of small clusters, there may be only one dedicated master node that serves as both the JobTracker and the NameNode.

When launching a MapReduce job, Hadoop first splits the input file into fixed-sized data blocks (64 MB by default) that are then stored in HDFS. The MapReduce job is divided into certain number of map and reduce tasks that can be run on slave nodes in parallel. Each map task processes one data block of the input file, and outputs intermediate key/value pairs generated by the user defined map function. The output of a map task is first written to a memory buffer, and then written to a spill file on local disk when the data in the buffer reaches certain threshold. All the spill files generated by one map task are eventually merged into one single partitioned and sorted intermediate file on the local disk of the map task. Each partition in this intermediate file is to be processed by one different reduce task, and is copied by the reduce task as soon as the partition becomes available. Running in parallel, reduce tasks then apply the user defined reduce function to the intermediate key/value pairs associated with each intermediate key, and generate the final output of the MapReduce job.

In a Hadoop cluster, the JobTracker is the job submission node where a client application submits the MapReduce job to be executed. The JobTracker organizes the whole execution process of the MapReduce job, and coordinates the running of all map and reduce tasks. TaskTrakers are the worker nodes which actually perform all the map and reduce tasks. Each TaskTraker has a configurable number of task slots for task assignment (two slots for map tasks and two for reduce tasks by default), so that the resources of a TaskTraker node can be fully utilized. The JobTracker is responsible for both job scheduling, i.e. how to schedule concurrent jobs from multiple users, and task assignment, i.e. how to assign tasks to all TaskTrackers. In this paper, we only address the problem of map task assignment. The map task assignment scheme of Hadoop adopts a heartbeat protocol. Each TaskTraker sends a heartbeat message to the JobTracker every few minutes to inform the latter that it's functioning properly and also whether it has an empty task slot. If a TaskTracker has an empty slot, the acknowledgment message from the JobTracker would contain information on the assignment of a new input data block.

To reduce the overhead of data transfer across network, the JobTracker attempts to enforce data locality when it performs task assignment. When a TaskTraker is available for task assignment, the JobTracker would first attempt to find an unprocessed data block that is located on the local disk of the TaskTracker. If it cannot find a local data block, the JobTracker would then attempt to find a data block that is located on certain node that is on the same rack as the TaskTracker. If it still cannot find a rack-local block, the JobTracker would finally find an unprocessed block that is as close to the TaskTracker as possible based on the topology information on the cluster.

While map tasks have only one stage, reduce tasks consist of three: copy, sort and reduce stages. In the copy stage, reduce tasks copy the intermediate data produced by map tasks. Each reduce task is usually responsible for processing the intermediate data associated with many intermediate keys. Therefore, in the sort stage, reduce tasks need to sort all the intermediate data copied by the intermediate keys. In the reduce stage, reduce tasks apply the user defined reduce function to the intermediate data associated with each intermediate key, and store the output in final output files. The output files are kept in the HDFS, and each reduce task generates exactly one output file.

Hadoop fault tolerance mechanisms

Failures are mostly inevitable when Hadoop runs at large scales. Consequently, Hadoop is designed as a fault-tolerant framework that can handle various failures with minimum impact on the quality of service. There are three different failure modes, task failure, TaskTracker failure, and JobTracker failure. When the TaskTracker detects a task failure, it will mark the task attempt as *failed*, free the task slot on which the task is running, and notify the JobTracker of the failure in its heartbeat message. The JobTracker will then try to reschedule execution of that task on a different TaskTracker. The whole job will fail, if any task fails a configurable number of times (four times by default), which usually means the user code is buggy. TaskTracker failure occurs, when the JobTracker hasn't received any heartbeat message from certain TaskTracker for a configurable period of time (10 minutes by default). TaskTracker failure is a much more serious failure mode than task failure, because the intermediate output of all map tasks that previously ran and finished on the failed TaskTracker becomes inaccessible. In this case, the JobTracker will rerun all those completed map tasks, and reschedule any tasks in progress on other TaskTrackers. JobTracker failure is the most serious failure mode, but it is not likely to happen as the chance that a particular machine fails is low. In the case of JobTracker failure, Hadoop provides a configuration option that can attempt to recover all jobs that were running at the time the failure occurred.

More detailed discussions on various aspects of MapReduce and Hadoop can be found in [8] and [9].

Issues with the Hadoop task assignment scheme

Both MapReduce and Hadoop were originally designed for computer clusters instead of computer clouds. Clusters are mostly a homogeneous computing environment, where homogeneous nodes run in similar load conditions, and tasks of the same type tend to start and finish at roughly close times. However, the situation is completely different in the clouds. Cloud service providers employ virtualization technology to abstract physical resources, simplify their usage, improve hardware utilization, and provide user isolation for security purposes. Although current virtualization technology can isolate CPU and memory usage effectively, co-located VM's still have to compete for both network and disk bandwidth, especially in the case of I/O intensive workload, such as the MapReduce jobs. Even in homogeneous environments, network bandwidth is often a bottleneck, and it is more precious in the clouds due to the employment of virtualization technology. Consequently, a cluster of VM's in the clouds are mostly heterogeneous instead of homogeneous, and there exist two different sources of heterogeneity. First, when the cloud service user only uses small to medium numbers of VM's, most of these VM's would probably reside on distinct physical hosts, and hence would not compete with each other for the I/O resources. However, these VM's still have to compete with varying numbers of VM's belonging to other users running different applications, and hence face resource contention of different intensities that could change during the whole time period of data processing. Secondly, the even worse scenario is when the user runs Hadoop at large scales, and hence large numbers of VM's are allocated. In this case, most of the VM's belonging to the same user would not be isolated to each other anymore, and would have to compete with each other for I/O resources, which results in much higher heterogeneity in the VM cluster than in the first scenario. As an example, Zaharia et al. [10] conducted large scale experiments on Amazon EC2 to measure its heterogeneity. Their experimental results indicated that the performance difference can be of a factor of about 1.4 in the first scenario, and up to 2.7 in the second scenario due to the heterogeneity of the computing platform. To evaluate the performance of our proposed task assignment scheme, the above scenarios were resembled in the simulation as the slightly and highly heterogeneous environments.

The Hadoop task assignment scheme is simple and intuitive. Whenever a task slot becomes available, the scheme assigns a data block to it. Initially all slots only consume local blocks. After a while, certain slots, most likely the faster ones, would run out of local blocks, and as the data processing approaches the end of the map phase, more and more slots would so. The challenging question of how to utilize these task slots arises. The Hadoop scheme simply assigns a remote block to the slot to prevent it from becoming idle, which may not be appropriate for two reasons. First, task slots can process local blocks much faster than they can remote ones. Although the local slots have to start the processing later than the remote ones that are immediately available, the local slots may still be able to finish earlier than the remote ones. Therefore, it may not be necessary to assign data blocks to remote slots, even if the overhead of fetching remote data blocks is acceptable. Secondly, the utilization of remote slots comes at a price that may be high enough to offset or even outweigh its benefits. This is because reduce tasks start copying intermediate data produced by map tasks as soon as the data becomes available. In most cases, the copy stage accounts for the majority of the execution time of reduce tasks, and requires large amount of data transfer across network. Therefore, after the first batch of map tasks finish, all the reduce tasks would be running and fetching remote data across network. At this point, the network bandwidth would become the most precious resource within the cluster environment, especially if it is a VM cluster running in a cloud. Any map tasks processing remote blocks would increase the contention for network bandwidth, and slow down the copy stage of all running reduce tasks. Moreover, a remote map task also competes for the local disk bandwidth with all running local tasks, as the remote task needs to read the data block located on the local disk. The competition would slow down both local and remote tasks.

The second issue with the Hadoop scheme is about data locality. One rule works well in the context of data-intensive computing is moving processing to data, which is adopted by both MapReduce and Hadoop. The Hadoop scheduler always attempts to schedule a map task on a node that has a copy of the data block to be processed first. If it could not make it, the scheduler would instead schedule the task as close to a copy of the data block as possible based on the network topology information on the computer cluster. Nodes in a computer cluster are typically connected by high performance network, which means the network topology of the cluster is known and remain unchanged throughout the whole processing period. However, in the case of clouds, the VM's in a cluster are linked together by certain network topology that is completely unknown or at least obscure. And, cloud operators employ the technology of VM migration to balance load among physical servers or bring down certain servers for maintenance purposes, which means the network topology of a cluster of VM's could change during the whole period of data processing. Therefore, when running in the clouds, the Hadoop scheduler would not have sufficient information on the network topology of the VM cluster. As a result, the scheduler may schedule a map task on a VM to process certain data block located on another VM that could be many network hops away from the first one. Since the data block must be fetched across the network to be processed, that single map task could seriously slow down the data processing if any part of the network between those two VM's is congested.

The completion time of MapReduce jobs is an important performance metric of Hadoop, especially for the use cases of ad-hoc queries where users need to obtain the results as quickly as possible, and for the use cases of public clouds where users are charged according to the amount of time the provisioned resources are used. The Hadoop scheme is reactive instead of proactive, it doesn't make any explicit effort to shorten the Map Phase Completion Time (MPCT). Whereas, the MPCT is crucial to the job completion time of MapReduce jobs, because reduce tasks need to copy intermediate data produced by map tasks before they can start processing, and typically all reduce tasks need the intermediate data produced by each and every map task including the one that finishes the last, which stops at the MPCT.

Related mathematical model

Minimum makespan scheduling is one classical combinatorial optimization problem, where given a set of jobs and a cluster of machines, the scheduling is to assign jobs to machines so that the makespan (maximum completion time of all jobs) is minimized. There exist many variants of this problem, but the one that is specifically related to the map task assignment problem is scheduling identical jobs on uniform parallel machines. The scheduling problem can be defined as follows. A set of identical and independent jobs J_i ($i = 1, 2, \ldots, n$) need to be assigned to a set of uniform machines M_j ($j = 1, 2, \ldots, m$) running in parallel. Machines are uniform if they can process at most one job at a time, and do so at known processing speeds, which can be either the same or different for different machines. The scheduling objective is to minimize the makespan.

In 1990, Dessouky et al. [11] proposed an algorithm for solving the above scheduling problem, which is based on the Earliest Completion Time (ECT) rule. The algorithm maintains a priority queue of the completion times of the *m* machines' next job assignment. It selects jobs in sequential order and schedules each job on the machine that can complete it the earliest among all machines. And the earliest completion time in the priority queue is replaced by the updated completion time of the machine that is assigned the job. The procedure continues until all *n* jobs are assigned, and returns a series of job completion times of $t_1, t_2, \ldots, t_n$, where $t_1 \le t_2 \le \ldots \le t_n$. It is obvious that no job would be assigned to a machine in such a manner that its completion time can be reduced by the assignment of the job to another machine. Therefore, the Minimality Property can be directly reasoned out from the algorithm procedure, which asserts that there does not exist any other schedule with job completion times $t_1' \le t_2' \le \ldots \le t_n'$, such that $t_k' < t_k$ for any $k = 1, 2, \ldots, n$. In other words, the completion time of each job is the earliest possible time.

In spite of its simplicity, the ECT algorithm is optimal, which can be proved by the Minimality Property. Suppose we have another schedule with job completion times $t_1', t_2', \ldots, t_n'$, which can yield a smaller makespan. Whether the series of $t_1', t_2', \ldots, t_n'$ is in certain order or no order at all, we can always sort it to make it in the ascending order. The last completion time in this sorted series is the makespan of the schedule, which cannot be possibly smaller than t_n (the makespan of the schedule produced by the ECT algorithm) according to the Minimality Property.

The ECT task assignment scheme

Although the map task assignment problem is related to scheduling identical jobs on uniform parallel machines, the former is not identical to the latter for two reasons. First, each machine in the latter can process all jobs at the same speed. Whereas, in the former, machines can process local blocks faster than they can remote ones. Secondly, each machine in the latter has known and constant processing speed, nevertheless the processing speeds of machines in the former are unknown and fluctuate over the whole processing period.

However, the ECT rule proposed by Dessouky et al. is still applicable in the map task assignment problem. Since all data blocks are of the same size, we assume all of them require the same amount of processing work to be processed. Therefore, in the context of map task assignment, the identical jobs are the input data blocks to be processed, and the machines are the task slots on VM's. The scheduler assigns a data block (job) to a task slot (machine) by scheduling a map task on that slot. The whole map phase of a MapReduce job can be considered as a multi-step process. The number of the steps is equal to the number of input data blocks to be processed. In each step, one single data block is processed by certain slot. If we can minimize the completion time of each step, we would be able to minimize the MPCT. Suppose we have a set of task slots S_i ($i \in \{1, 2, \ldots, m\}$) for the processing of input data blocks. Each task slot S_i has an available time T_i at which it would complete its current data block and be available to process its next data block. And if we know that it will take a task slot processing time P_i ($i \in \{1, 2, \ldots, m\}$) to process its next data block, then the completion time of its next data block will be $C_i = T_i + P_i$ ($i \in \{1, 2, \ldots, m\}$). The minimum value $C_j = \min\{C_i\}$ ($i, j \in \{1, 2, \ldots, m\}$) is the earliest possible completion time of the next step of the whole map phase, which means we can minimize the completion time of the next step by assigning a data block to the task slot S_j ($j \in \{1, 2, \ldots, m\}$). Therefore, the basic task assignment strategy of our new scheme is the ECT rule, i.e. the slot that can complete a data block (either a local or a REMOTE one) the earliest among all slots is assigned one. Note that this task slot doesn't have to be the earliest available slot, to which the Hadoop scheme would always assign the next data block. We call the new scheme ECT as it's based on the ECT rule. Figure 1 is a flowchart of the ECT scheme. The two concurrent processes: {When a slot becomes available, update its PTE} and {Initialize the priority queue of completion times} are modeled in the flowchart by a concurrent construct similar to the Fork symbol used in UML activity diagrams.

Since the processing speeds of slots fluctuate, ECT predicts the amount of time it takes a slot to process a data block by sampling the processing behaviors of that slot and averaging those samples into a Processing Time Estimate (PTE). For each slot, there are two types of PTE's, local and remote PTE's for the processing of local and remote blocks, respectively. In the simulation, the processing times of a slot were randomly generated within certain range, and hence the PTE was simply computed as the average of all processing time samples. Nevertheless, in practice, the PTE can also be calculated in a way

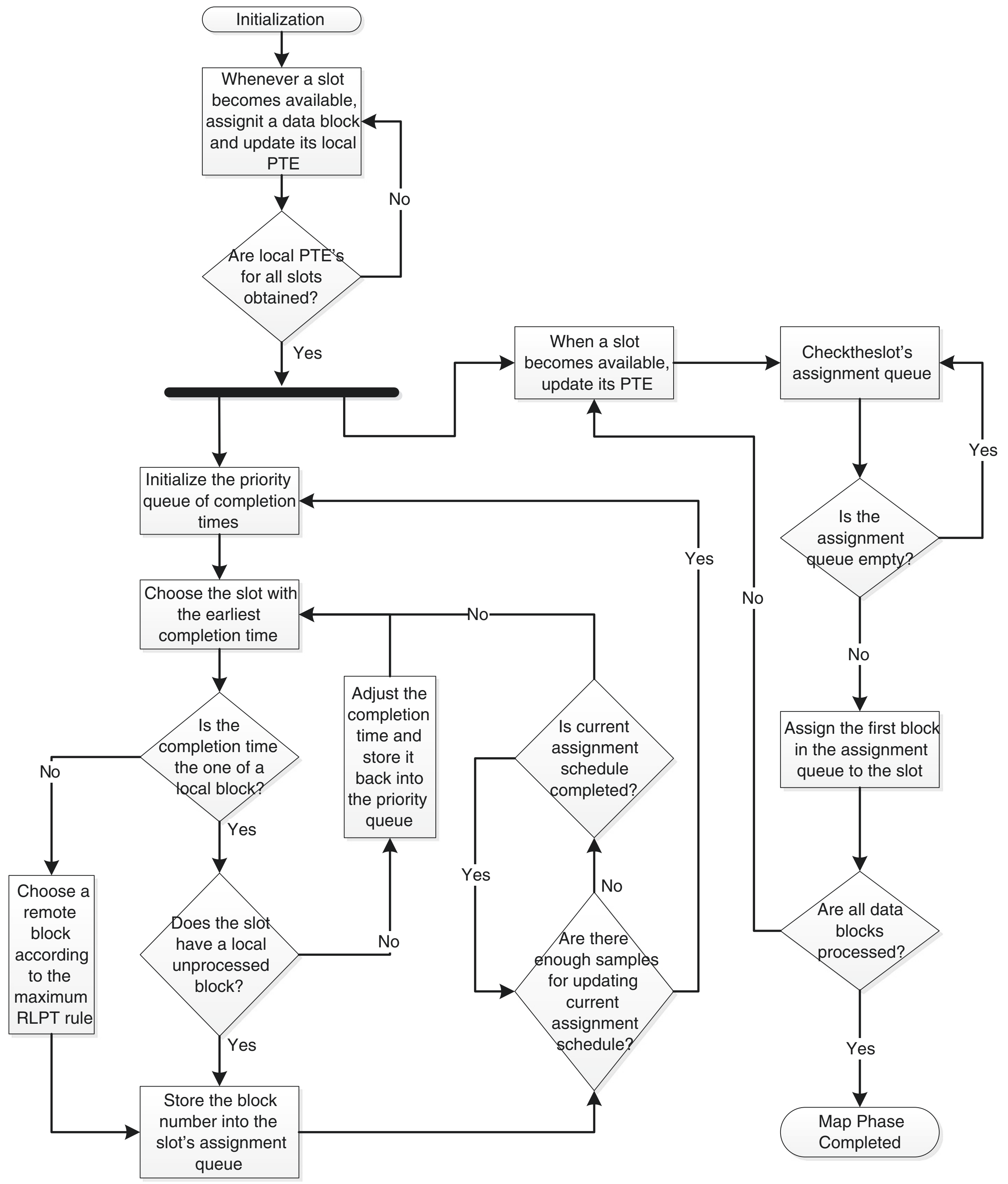
Initialization
Whenever a slot becomes available, assignit a data block and update its local PTE
Are local PTE's for all slots obtained?
No
Yes
When a slot becomes available, update its PTE
Checktheslot's assignment queue
Is the assignment queue empty?
Yes
No
Assign the first block in the assignment queue to the slot
Are all data blocks processed?
No
Yes
Map Phase Completed
Initialize the priority queue of completion times
Yes
Choose the slot with the earliest completion time
No
Is the completion time the one of a local block?
No
Yes
Adjust the completion time and store it back into the priority queue
Is current assignment schedule completed?
Choose a remote block according to the maximum RLPT rule
Does the slot have a local unprocessed block?
No
Yes
Are there enough samples for updating current assignment schedule?
No
Yes
Store the block number into the slot's assignment queue

Figure 1 Flowchart of ECT.

that is similar to the way the round-trip time is estimated in network transport protocols to allow the PTE adapt to sample variance more quickly. Whenever a new sample is obtained, ECT can update the task slot's PTE according to the following formulas:

$$\mathrm{PTE}_1 = \mathrm{S}_1$$

$\mathrm{PTE}_{j+1} = \alpha \times \mathrm{PTE}_j + (1-\alpha)\mathrm{S}_{j+1} \quad (j = 1, 2, 3, \ldots),$ where PTE_j is the current PTE, PTE_{j+1} is the new estimate based on the new sample S_{j+1}, and α is a constant between 0 and 1 that controls how fast the PTE adapts to sample variance. The value of α can be set to higher ones for more variable computing environments, such as virtualized clouds, and lower ones for more stable computing environments, such as real clusters.

Before it can assign data blocks according to the ECT rule, ECT needs to obtain the local PTE of all task slots. Therefore, ECT assigns data blocks to slots whenever they become available in its first stage. After the local PTE is available for each slot, ECT starts assigning data blocks according to the ECT rule by working out an assignment schedule. ECT maintains a priority queue of the completion times of all slots' next data block based on their PTE's. At each step of the schedule calculation, ECT chooses the slot that can complete its next data block the earliest among all slots. The next data block can be either local or remote to the slot. Initially, all completion times in the priority queue are the ones of local blocks, as all slots would process local blocks first. At certain point, each slot would run out of its local blocks. The situation that needs special treatment is when there are no remaining local blocks to be assigned to certain slot that has the earliest completion time in the priority queue. Instead of assigning a remote block to the slot immediately, ECT needs to replace the slot's completion time in the priority queue, which is the completion time of its next LOCAL block, with the completion time of its next REMOTE block. Only when this updated completion time shows up as the earliest time in the priority queue, ECT will then assign a remote block to the corresponding slot. And, this remote assignment still minimizes the completion time of the corresponding processing step.

When it does need to assign a data block to a remote slot, ECT chooses a block on the local disk of the slot that has the maximum Remaining Local Processing Time (RLPT) to further reduce the amount of remote processing. The RLPT of a slot is calculated as the number of unprocessed local blocks times the current local PTE of the slot. For the slot having maximum RLPT, the assignment of its local blocks to remote slots is less likely (compared with other slots) to make it run out of local blocks before the map phase ends, and hence it is less likely this slot would engage in remote processing at a later time.

ECT maintains an assignment queue for each slot to store the block numbers of all data blocks assigned to that slot. Task slots don't wait until the whole assignment scheme is calculated to receive their assignments. Instead, as soon as the first data block is assigned to a slot, the slot can start processing the data block. And a slot's data processing would not be interrupted, as long as its assignment queue is not empty.

The processing speed of any task slot always fluctuates, and ECT updates each slot's PTE whenever a new sample becomes available for that slot. ECT also updates the assignment schedule of data blocks to make it better reflect the changing computing environment. After a configurable number of processing time samples are obtained, ECT will work out a new assignment schedule to replace the current one. The update of assignment schedule also helps reduce the effect of estimation error, which is the difference between the PTE and the actual slot processing time. Estimation error will accumulate in the calculation of an assignment schedule, therefore the accuracy of the schedule decreases from the beginning to the end. If the schedule is frequently updated, only the beginning part of it, which is more accurate, will actually be executed.

Evaluation

We evaluated the performance of ECT compared with the Hadoop scheme by extensive simulation. We used Discrete-Event Simulation to model the operation of the map phase under both the Hadoop scheme and ECT. The discrete sequence of events is the completion of individual data blocks, each of which is completed at a particular instant in time and causes the change of state in the simulated map phase operation. After the state of the map phase operation has been updated, the current simulation time skips to the completion time of the data block that is to be completed next. The simulation procedure continues until all data blocks are processed. Various statistics are recorded during the simulation, and the ones of special interest are the MPCT, and the total Number of data Blocks Remotely Processed (NBRP). The simulation program also records the completion time of each data block for the generation of processing time traces. The common settings of all simulation scenarios are shown in Table 1.

We tested ECT in two different computing environments that were typical in the public clouds, the slightly and highly heterogeneous environments. To resemble the heterogeneous environment in the clouds, we assumed the base processing speeds of all 1000 task slots were evenly distributed within a fixed range, while the two slots on the same VM had identical base speed. In the simulation, we

Table 1 Common settings of all simulation scenarios

Total processing workload of input data	100,000 task slot × time units
Number of VM's	500
Number of map task slots on each VM	2 (Hadoop Default)
Duplication factor of data blocks	3 (Hadoop Default)
Speculation	No

actually used slot processing times to represent slot processing speeds. There were two types of slot processing times, Local Processing Time (LPT) and Remote Processing Time (RPT), which indicated the amount of time it took a slot to process a local and remote block, respectively. LPT was randomly generated within the range [(1-VPPT)t, (1 + VPPT)t], where t was the base processing time of the task slot, and VPPT was the Variation Percentage of Processing Time, which was used to reflect the fact that the actual slot processing times fluctuated during the whole processing period. Most MapReduce jobs belong to the relatively short and interactive category, so their job completion times are usually measured in minutes instead of hours. As a result, it is unlikely that the actual slot processing times would fluctuate significantly during the whole processing period. Therefore, we ran the simulation at three different VPPT values: 2.5%, 5% and 10%. The base processing time t's of slots on different VM's were assumed to be evenly distributed within the range [(1-P)T, (1 + P)T], where T = 100,000 / total number of data blocks. In the simulation, the value of P was set to 0.2 and 0.5 to resemble the slightly and highly heterogeneous environments respectively, which was based on the experimental results obtained on Amazon EC2 by Zaharia et al. [10]. The RPT consisted of two parts, the LPT and the amount of time it took the processing slot to fetch the data bock across the network. In the simulation, the RPT was calculated as (RPC × LPT), where RPC was the Remote Processing Coefficient used to reflect the overhead of fetching remote data blocks and hence was greater than one. Since LPT was randomly generated within the range [(1-VPPT)t, (1 + VPPT)t], RPT was randomly generated within the range [RPC(1-VPPT)t, RPC(1 + VPPT)t].

For both slightly and highly heterogeneous environments, we ran the simulation at four typical combinations of RPC and VPPT values as shown in Table 2. Scenario one resembles the computing environment where the remote fetching overhead is low and background load on VM's is fairly stable. Scenario two resembles the environment where the remote fetching overhead is medium and background load is relatively stable. Whereas, in Scenario three, task slots experience network congestion (disk contention), and slot processing speeds fluctuate. Scenario four is to evaluate the performance of ECT in the circumstance where there is no overhead of fetching remote data blocks.

Table 2 RPC and VPPT settings in the simulation

	Remote processing coefficient (RPC)	Variation percentage of processing time (VPPT)
Scenario One	1.5	± 2.5%
Scenario Two	2.5	± 5%
Scenario Three	4.0	± 10%
Scenario Four	1.0	± 2.5%

For the performance comparison, we examined two metrics: the MPCT and the NBRP. Although minimum MPCT is one of the ultimate goals of all task assignment schemes, the NBRP is also an important metric in the sense that smaller NBRP values mean less remote processing employed, and thus it is less likely the data processing would be delayed by network congestion and/or disk contention. Furthermore, remote processing involves more factors than local processing, thus it is more likely to fail due to its complexity. Therefore, a task assignment scheme that employs less remote processing would be more favorable to one that employs more, if the MPCT's achieved by both schemes are close.

Slightly heterogeneous environment

For the slightly heterogeneous environment, it can be reasoned out from the simulation settings that the shortest possible processing time of one data block is 0.9 × 0.8 T = 0.72 T time units, and the longest possible processing time 1.1 × 1.2 T = 1.32 T time units, which is less than two times the shortest time. This means at the time the slowest slot finishes its first block and is assigned its second block, all other slots would be processing their second blocks. Consequently, ECT needs to assign the first 2000 data blocks to obtain the processing time estimates of all task slots, before it can assign the remaining data blocks according to the ECT rule. Therefore, for each of the four scenarios, we ran the simulation with the total number of data blocks taking on values of 2500, 3000, 3500, … , 8000. The amount of processing work of one data block was adjusted according to the total number of data blocks (i.e. T = 100,000/total number of data blocks), so that the results of different total numbers of data blocks are comparable to each other.

Figures 2, 3 and 4 present the MPCT's achieved by both schemes at different total numbers of data blocks in scenarios one, two and three, respectively. It can be observed that ECT always achieves less MPCT than the Hadoop scheme, and the reduction is most significant in scenario three, where the remote fetching overhead is high. Simulation results of different total numbers of

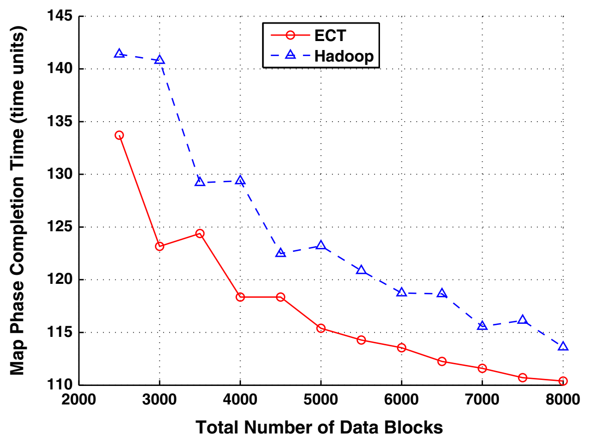

Figure 2 Map phase completion times of scenario one. (Slightly heterogeneous environment)

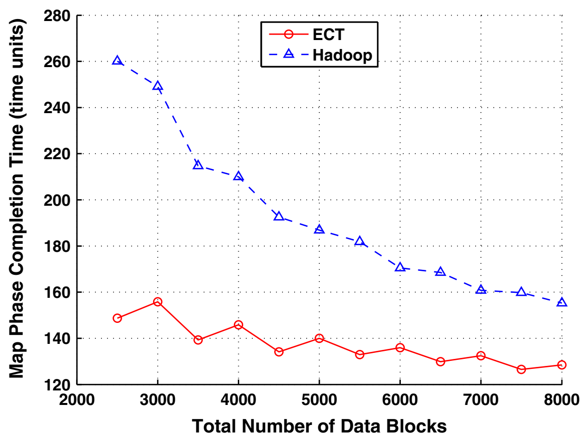

Figure 4 Map phase completion times of scenario three. (Slightly heterogeneous environment)

data blocks in scenarios one, two and three are shown in Tables 3, 4 and 5, respectively. (Only partial results are included due to length limitation. All results are the average of ten simulation runs.) The average results of ALL different total numbers of data blocks for each scenario is shown in Table 6, which indicate that the average MPCT achieved by ECT is 5.6%, 15.3% and 28.6% less than the one achieved by the Hadoop scheme, and the average NBRP under ECT is 29.5%, 73.8% and 97.5% less than the one under the Hadoop scheme in scenarios one, two and three, respectively. The MPCT reduction of ECT is mostly attributed to its capability to reduce remote processing, which could seriously impair the MPCT performance. As mentioned earlier, ECT is designed based on an optimal algorithm, which assumes that all slot processing speeds remain the same throughout the whole processing period. If this assumption held, ECT would be able to yield the minimum possible MPCT. Unfortunately, in the case of map task assignment, the slot processing speeds always fluctuate, and hence the optimal solution does not actually exist. However, since the remote fetching overhead is relatively high compared with the fluctuation of the slot processing times, ECT can still effectively reject unnecessary remote processing despite the estimation error of processing time.

It can also be observed from the simulation results that ECT is much more robust to network congestion and/or disk contention than the Hadoop scheme. As shown in Table 6, when the RPC increases from 1.5 to 4.0, the average MPCT of Hadoop rises remarkably from 124.2 to 192.5 time units (a 55.0% increase), whereas the average MPCT of ECT only rises from 117.2 to 137.5 time units (a 17.3% increase). The robustness of ECT is a result of its capability to reduce the NBRP accordingly when the remote fetching overhead rises. As shown in Table 6, when the RPC increases from 1.5 to 4.0, the average NBRP under ECT drops sharply from 226.6 to 7.5 (a 96.7% decrease), whereas the average NBRP under Hadoop only decreases slightly from 321.5 to 306.2 (a 4.8% decrease). As mentioned earlier, ECT projects and sorts all task slots' completion times of their next data block and assigns blocks to slots at the sorted order, from the earliest to the latest. When the remote fetching overhead increases, it becomes more and more unlikely that a data block would be assigned to a remote slot. In contrast, the Hadoop scheme doesn't consider either the remote fetching overhead or the MPCT when assigning data blocks. The decrease of the NBRP under Hadoop is actually related to the VPPT instead of the RPC, because when the fluctuation of slot processing times increases, the whole cluster of 1000 task slots becomes slightly less heterogeneous due to the fact that the base slot processing times are evenly distributed within a fixed range.

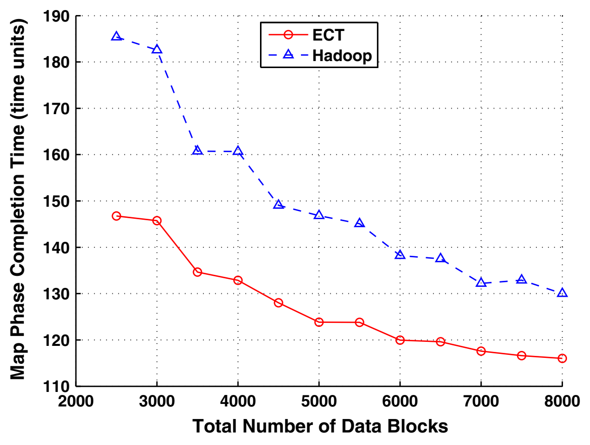

Figure 3 Map phase completion times of scenario two. (Slightly heterogeneous environment)

Table 3 Simulation results of scenario one (Slightly heterogeneous environment, low remote fetching overhead, and stable slot processing speeds)

RPC = 1.5	Total number of data blocks						
VPPT = ± 2.5%	2500	3000	4000	5000	6000	7000	8000
Number of blocks assigned according to the ECT rule	500	1000	2000	3000	4000	5000	6000
MPCT of Hadoop (time units)	141.4	140.8	129.4	123.2	118.7	115.6	113.6
MPCT of ECT (time units)	133.7	123.2	118.4	115.4	113.5	111.6	110.4
ECT reduction in MPCT	5.4%	12.5%	8.5%	6.3%	4.4%	3.4%	2.8%
NBRP under Hadoop	297.3	85.7	222.4	315.2	375.5	413.9	449.9
NBRP under ECT	159.4	56.4	143.5	214.7	265.7	300.4	325.6
ECT reduction in NBRP	46.4%	34.2%	35.5%	31.9%	29.2%	27.4%	27.6%

Consequently, the average NBRP under Hadoop decreases insignificantly.

Another important observation from the simulation results is the total number of data blocks has a significant impact on the MPCT's achieved by both schemes. It can be seen from Figures 2, 3 and 4, in general, the MPCT decreases while the total number of data blocks increases for both schemes. The data processing proceeds at the highest speed before it enters the ending stage, because all slots run in parallel to process data blocks. After it enters the ending stage, the processing proceeds slower and slower, as more and more slots stop running. All slots would eventually stop and mostly they stop at different times. In general, the smaller the data blocks, the closer the stop times of different slots. When the total number of data blocks increases, the size of them decreases accordingly, thus overall the stop times of different slots get closer, which has the same effect as increasing the average speed of the data processing in the ending stage and hence decreases the MPCT. Although they can yield shorter MPCT's, larger values of the total number of data blocks will increase the amount of time it takes to duplicate the data blocks and distribute all the copies to different VM's, and will also increase the maintenance overhead of the Hadoop Distributed File System and the task assignment overhead of the JobTracker in a Hadoop cluster, which necessitates a wise tradeoff between performance and overhead. Moreover, as the data blocks become smaller, the distinction of the MPCT performance between different task assignment schemes also gets smaller, because in general a bad decision on the assignment of a data block would increase the MPCT less than it would when the data block is bigger. Consequently, the MPCT reduction of ECT over the Hadoop scheme decreases when the total number of data blocks increases, as shown in Figures 2, 3 and 4 where the curves of both schemes approach to each other while stretching to the right.

The processing time traces of one typical case are shown in Figure 5. It can be seen that the time traces of both schemes match perfectly except for the tail part. This is because both schemes keep all slots running in parallel until the ending stage, when there are not enough remaining data blocks for the schemes to do so. And both schemes have fairly close block completion times as simulation results are the average of ten simulation runs. The two time traces diverge at the tail. While the ECT trace rises in slightly accelerated rate, the Hadoop trace rises sharply. Since there are less and less slots running in the ending stage, the data processing gradually slows down, which causes the accelerated rising of the ECT trace. On the other hand, the Hadoop scheme

Table 4 Simulation results of scenario two (Slightly heterogeneous environment, medium remote fetching overhead, and relatively stable slot processing speeds)

RPC = 2.5	Total number of data blocks						
VPPT = ± 5%	2500	3000	4000	5000	6000	7000	8000
Number of blocks assigned according to the ECT rule	500	1000	2000	3000	4000	5000	6000
MPCT of Hadoop (time units)	185.4	182.6	160.7	146.8	138.2	132.2	130.0
MPCT of ECT (time units)	146.8	145.7	132.9	123.8	120.0	117.6	116.0
ECT reduction in MPCT	20.8%	20.2%	17.3%	15.6%	13.2%	11.1%	10.8%
NBRP under Hadoop	285.2	89.6	214.1	306.9	365.5	411.2	443.1
NBRP under ECT	0.9	15.0	23.6	36.4	85.0	135.6	176.6
ECT reduction in NBRP	99.7%	83.3%	89.0%	88.1%	76.7%	67.0%	60.1%

Table 5 Simulation results of scenario three (Slightly heterogeneous environment, high remote fetching overhead, and less stable slot processing speeds)

RPC = 4.0	Total number of data blocks						
VPPT = ± 10%	2500	3000	4000	5000	6000	7000	8000
Number of blocks assigned according to the ECT rule	500	1000	2000	3000	4000	5000	6000
MPCT of Hadoop (time units)	260.1	249.1	210.0	186.9	170.5	160.8	155.3
MPCT of ECT (time units)	148.7	155.8	145.9	140.0	135.9	132.5	128.4
ECT reduction in MPCT	42.8%	37.4%	30.5%	25.1%	20.3%	17.6%	17.3%
NBRP under Hadoop	260.4	95.4	204.8	298.7	358.3	402.1	437.8
NBRP under ECT	0	0	0.1	1.7	6.6	14.6	26.6
ECT reduction in NBRP	100%	100%	100%	99.4%	98.2%	96.4%	93.9%

works reasonably well until most slots run out of local blocks. From this point on until the end of the map phase, the majority of the data blocks would be processed by remote slots, and large amount of remote processing causes the sharp rise of the Hadoop trace.

Table 6 also includes the standard deviations of all simulation results in addition to the averages. It can be observed that, in scenarios one, two, and three where there exists remote fetching overhead, the variance of ECT results is always lower than the one of Hadoop results for both MPCT and NBRP, and the higher the overhead, the bigger the difference. The most significant difference occurs in scenario three, where the standard deviation of ECT MPCT is 8.8 time units compared with Hadoop's 34.7 time units, and the standard deviation of ECT NBRP is 10.9 compared with Hadoop's 96.4. This is because ECT has the capability to automatically adapt to the congestion (contention) level of the cluster network (VM disks), which the Hadoop scheme doesn't have. When the remote fetching overhead is sufficiently high, remote processing of data blocks in the ending stage is mostly rejected by ECT. As shown in Table 5, compared with the Hadoop scheme, ECT only allows very limited (if any) number of data blocks to be processed remotely at different total numbers of data blocks in scenario three, which causes much more stable NBRP and hence much more stable MPCT. The benefit of stable NBRP and MPCT is that better performance can be achieved by splitting the input file into less number of data blocks, and hence incurring less overhead.

Highly heterogeneous environment

For the highly heterogeneous environment, it can be reasoned out from the simulation settings that the shortest possible processing time of one data block is 0.9 × 0.5 T = 0.45 T time units, and the longest possible processing time 1.1 × 1.5 T = 1.65 T time units, which is larger than three times the shortest time. As a result, ECT needs to assign more data blocks during its first stage to obtain the processing time estimates of all slots than it does in the slightly heterogeneous environment. Therefore, for

Table 6 Simulation results of slightly heterogeneous environment

	Scenario			
	One	Two	Three	Four
RPC	1.5	2.5	4.0	1.0
VPPT	± 2.5%	± 5%	± 10%	± 2.5%
Average of Hadoop MPCT (time units)	124.2	150.1	192.5	111.9
Standard deviation of Hadoop MPCT (time units)	9.3	18.8	34.7	4.9
Average of ECT MPCT (time units)	117.2	127.1	137.5	110.4
Standard deviation of ECT MPCT (time units)	7.0	10.8	8.8	5.1
ECT reduction in average MPCT	5.6%	15.3%	28.6%	1.4%
Average NBRP under Hadoop	321.5	314.4	306.2	—
Standard deviation of NBRP under Hadoop	99.4	97.4	96.4	—
Average NBRP under ECT	226.6	82.4	7.5	—
Standard deviation of NBRP under ECT	80.7	60.2	10.9	—
ECT reduction in average NBRP	29.5%	73.8%	97.5%	—

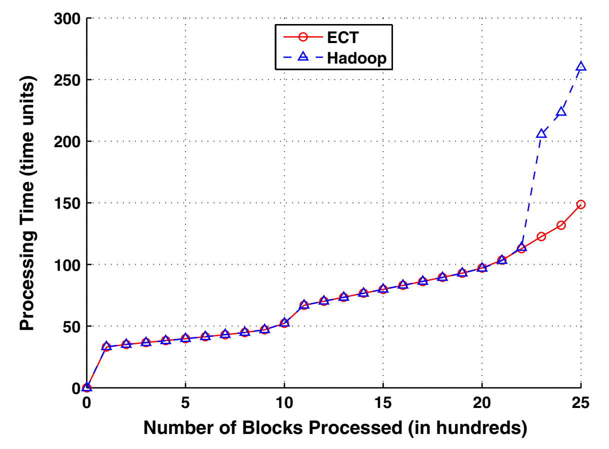

Figure 5 Processing time traces of scenario three. (Slightly heterogeneous environment, Total Number of Data Blocks = 2500)

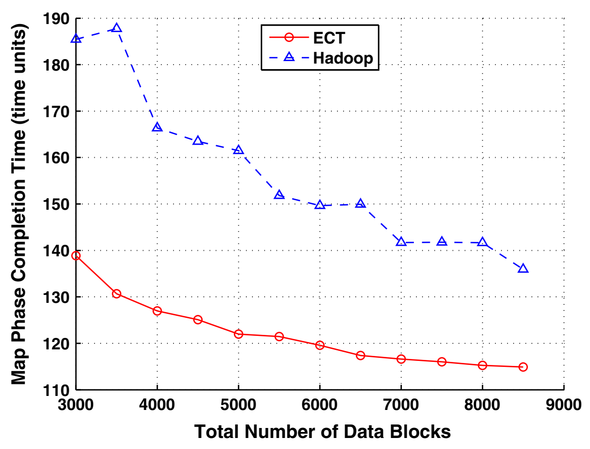

Figure 7 Map phase completion times of scenario two. (Highly heterogeneous environment)

each of the four scenarios, we ran the simulation with the total number of data blocks taking on values of 3000, 3500, 4000, ... , 8500, which were slightly larger than the values in the slightly heterogeneous environment.

Figures 6, 7 and 8 present the MPCT's achieved by both schemes at different total numbers of data blocks in scenarios one, two and three, respectively. Simulation results of different total numbers of data blocks in scenarios one, two and three are shown in Tables 7, 8 and 9, respectively. (Only partial results are included due to length limitation. All results are the average of ten simulation runs.) The average results of ALL different total numbers of data blocks for each scenario is shown in Table 10, which indicate that the average MPCT achieved by ECT is 12.2%, 22.0% and 33.7% less than the one achieved by the Hadoop scheme, and the average NBRP under ECT is 11.4%, 31.4% and 53.7% less than the one under the Hadoop scheme in scenarios one, two and three, respectively. Even in scenario four, where the remote fetching overhead is zero, the average MPCT achieved by ECT is still 7.0% less than the one achieved by the Hadoop scheme. The MPCT reduction in this case is solely attributed to the ECT rule, which can yield better results than the simple Hadoop strategy even when the slot processing speeds fluctuate over time. And, the stabler the slot processing speeds, the less the MPCT achieved by ECT is expected to be due to the optimality of the ECT rule.

When comparing the results in Table 10 with the ones in Table 6, we can see that both schemes assigned more data blocks to remote slots in the highly heterogeneous environment due to the increased heterogeneity. As a result, the average MPCT of the Hadoop scheme increases slightly in scenarios one, two and three, where there exists remote fetching overhead. There is no remote fetching overhead in scenario four, hence the faster slots would get assigned more remote blocks than they would

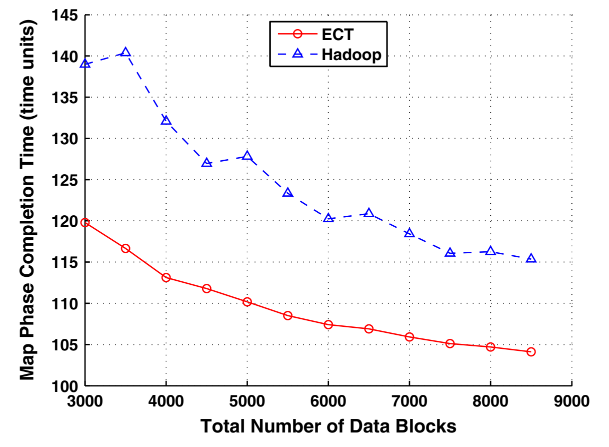

Figure 6 Map phase completion times of scenario one. (Highly heterogeneous environment)

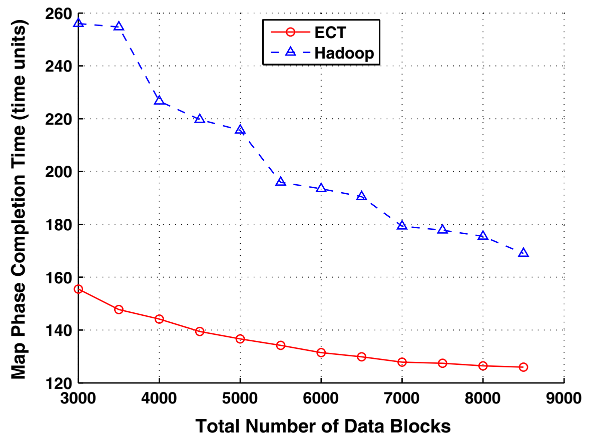

Figure 8 Map phase completion times of scenario three. (Highly heterogeneous environment)

Table 7 Simulation results of scenario one (Highly heterogeneous environment, low remote fetching overhead, and stable slot processing speeds)

RPC = 1.5	**Total number of data blocks**						
VPPT = ± 2.5%	**3000**	**3500**	**4500**	**5500**	**6500**	**7500**	**8500**
Number of blocks assigned according to the ECT rule	725.1	1225.5	2226.0	3224.3	4225.7	5227.8	6224.3
MPCT of Hadoop (time units)	139.0	140.4	127.0	123.3	120.9	116.1	115.3
MPCT of ECT (time units)	119.8	116.7	111.8	108.5	106.9	105.1	104.1
ECT reduction in MPCT	13.8%	16.9%	11.9%	12.0%	11.6%	9.4%	9.7%
NBRP under Hadoop	435.3	468.4	631.3	733.7	883.4	1030.4	1145.0
NBRP under ECT	394.0	459.0	558.3	664.0	780.4	879.2	993.6
ECT reduction in NBRP	9.5%	2.0%	11.6%	9.5%	11.7%	14.7%	13.2%

in the first three scenarios. In the ending stage of scenario four, most running slots would be those faster ones processing remote blocks. Since the faster slots in the highly heterogeneous environment are faster than the ones in the slightly heterogeneous environment due to the simulation settings, the average MPCT of the Hadoop scheme in the former is less. On the other hand, ECT achieves smaller MPCT's in all four scenarios in the highly heterogeneous environment than it does in the slightly heterogeneous environment. This is because, in the former, the slot base processing time t's are evenly distributed within a wider range compared with the latter, whereas the VPPT still takes on the same values in all four scenarios. As a result, the distinction of slot processing times in the former is larger. Consequently, the estimation error of slot processing time impairs the optimality of the ECT rule to a less extent when ECT projects and compares the completion times of different slots, which yields less MPCT's.

Simulation results of the highly heterogeneous environment confirm again that ECT is much more robust to network congestion (disk contention) than the Hadoop scheme. As shown in Table 10, when RPC increases from 1.5 to 4.0, the average NBRP under ECT drops sharply from 691.6 to 287.7 (a 58.4% decrease), and the average MPCT achieved by ECT rises from 109.5 to 135.6 time units (a 23.8% increase). In contrast, the average NBRP under Hadoop only decreases from 780.6 to 621.8 (a 20.3% decrease), and the average MPCT achieved by Hadoop increases considerably from 124.7 to 204.5 time units (a 64.0% increase). The processing time traces of one typical case are shown in Figure 9, which exhibit the similar pattern as the ones in Figure 5 due to the same reason discussed in the previous section. Table 10 also includes the standard deviations of all simulation results in addition to the averages, which indicate again that the ECT results are more stable than the Hadoop results.

ECT limitations

The proposed ECT scheme has its limitations. First, the performance improvement of MPCT decreases when the total number of data blocks increases for the reason explained earlier. Secondly, the performance improvement of MPCT decreases when the remote fetching overhead decreases. As discussed earlier, after most task slots run out of local blocks, the Hadoop scheme will incur large amount of remote processing which accounts for most part of the MPCT performance difference between ECT and the Hadoop scheme. Therefore, when the remote fetching overhead decreases, the performance difference decreases as well. Finally, the performance improvement of NBRP decreases when the remote fetching overhead decreases. This is because ECT will assign more data blocks

Table 8 Simulation results of scenario two (Highly heterogeneous environment, medium remote fetching overhead, and relatively stable slot processing speeds)

RPC = 2.5	**Total number of data blocks**						
VPPT = ± 5%	**3000**	**3500**	**4500**	**5500**	**6500**	**7500**	**8500**
Number of blocks assigned according to the ECT rule	700.1	1199.5	2198.2	3200.3	4196.4	5200.1	6195.4
MPCT of Hadoop (time units)	185.4	187.8	163.5	151.8	149.9	141.8	135.9
MPCT of ECT (time units)	138.9	130.7	125.1	121.4	117.4	116.0	114.9
ECT reduction in MPCT	25.1%	30.4%	23.5%	20.0%	21.7%	18.2%	15.5%
NBRP under Hadoop	423.6	427.7	571.7	665.9	742.7	878.7	984.0
NBRP under ECT	241.2	273.3	378.4	455.3	527.5	625.1	699.8
ECT reduction in NBRP	43.1%	36.1%	33.8%	31.6%	29.0%	28.9%	28.9%

Table 9 Simulation results of scenario three (Highly heterogeneous environment, high remote fetching overhead, and less stable slot processing speeds)

RPC = 4.0	Total number of data blocks						
VPPT = ± 10%	3000	3500	4500	5500	6500	7500	8500
Number of blocks assigned according to the ECT rule	645.2	1140.3	2138.2	3142.3	4140.6	5136.1	6139.9
MPCT of Hadoop (time units)	256.0	254.7	219.7	195.9	190.5	177.8	169.0
MPCT of ECT (time units)	155.5	147.8	139.5	134.2	129.9	127.4	125.9
ECT reduction in MPCT	39.3%	42.0%	36.5%	31.5%	31.8%	28.3%	25.5%
NBRP under Hadoop	417.9	418.9	536.8	607.6	672.7	758.4	845.8
NBRP under ECT	119.1	151.2	206.7	268.4	340.5	400.7	457.3
ECT reduction in NBRP	71.5%	63.9%	61.5%	55.8%	49.4%	47.2%	45.9%

to remote task slots when the remote fetching overhead decreases. Although the Hadoop scheme will do so as well, ECT is much more sensitive to the remote fetching overhead, and hence will assign more data blocks than the Hadoop scheme.

Related work

Dean et al. introduced the MapReduce programming model, implementation details, and various refinements in [2]. Their work served as the fundamental basis for the development of Hadoop, as well as all the following research on both MapReduce and Hadoop.

Jiang et al. presented a comprehensive performance study of Hadoop on Amazon EC2 in [12]. They identified certain design factors of Hadoop and discussed alternative methods for these factors. Their study indicated that the performance of Hadoop could be remarkably improved by tuning the design factors in a correct way. Lee et al. presented a comprehensive survey on MapReduce in [13]. They discussed the merits and drawbacks of MapReduce, various improvement contributions in literature, and remaining open issues regarding parallel processing with MapReduce. Vijayalakshmi et al. introduced various implementations of MapReduce in [14] including Hadoop. They evaluated and compared the performance of different implementations. The insights and experimental results provided in the above papers are very helpful to the in-depth understanding of Hadoop as well as the further improvement on the framework.

Zaharia et al. [10] focused on the speculative execution mechanism of Hadoop to reduce the job completion time. They discussed all the Hadoop assumptions related to speculative execution, and explained why these assumptions broke down in the clouds. They suggested a new strategy of speculative execution, which always speculatively executed the task that was predicted to finish the farthest into the future. The strategy launched a speculative copy of any potential straggler tasks, before they

Table 10 Simulation results of highly heterogeneous environment

	Scenario			
	One	Two	Three	Four
RPC	1.5	2.5	4.0	1.0
VPPT	± 2.5%	± 5%	± 10%	± 2.5%
Average of Hadoop MPCT (time units)	124.7	156.4	204.5	108.1
Standard deviation of Hadoop MPCT (time units)	8.7	17.0	29.9	5.3
Average of ECT MPCT (time units)	109.5	122.1	135.6	100.5
Standard deviation of ECT MPCT (time units)	5.0	7.3	9.5	3.4
ECT reduction in average MPCT	12.2%	22.0%	33.7%	7.0%
Average NBRP under Hadoop	780.6	685.1	621.8	—
Standard deviation of NBRP under Hadoop	242.3	189.3	145.7	—
Average NBRP under ECT	691.6	469.7	287.7	—
Standard deviation of NBRP under ECT	195.4	151.6	113.1	—
ECT Reduction in Average NBRP	11.4%	31.4%	53.7%	—

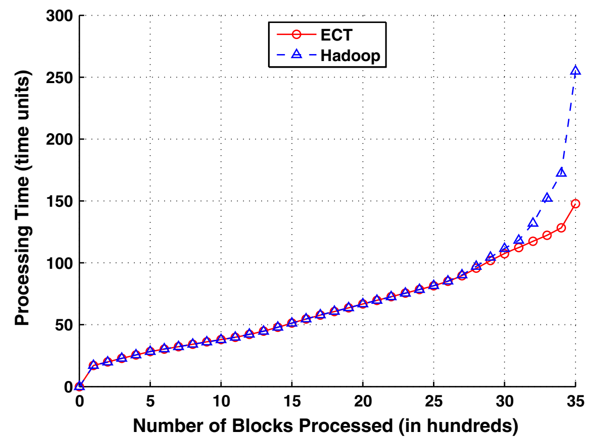

Figure 9 Processing time traces of scenario three. (Highly heterogeneous environment, Total Number of Data Blocks = 3500).

could actually prolong the completion of the whole MapReduce job at the expense of extra resource expenses. Whereas ECT works in a different way: it strives to minimize the completion time of each processing step of the map phase without any duplicated execution.

Pinedo discussed various deterministic scheduling models in the first part of [15], including the model of uniform parallel machines that was related to the map task assignment problem. Lawler et al. [16] pointed out that the problem of scheduling identical jobs on uniform parallel machines could be formulated as linear assignment problem and solved accordingly in polynomial time. Dessouky et al. [11] proposed a priority queue procedure for solving the same problem, which was an optimal and more efficient algorithm. The algorithm served as the basis of our improved task assignment scheme.

Conclusion

In this paper, we discussed the issues with the Hadoop task assignment scheme when Hadoop running in the clouds. We presented an improved scheme ECT based on an optimal algorithm for a related deterministic scheduling problem. We further conducted extensive simulation to evaluate the performance of ECT compared with the Hadoop scheme. The simulation results confirmed that ECT could significantly outperform the Hadoop scheme with respect to both the completion time of map phase and the amount of remote processing employed.

In future research, we plan to continue to address the map task assignment problem based on other related scheduling models. Also, we only focused on the task assignment aspect of Hadoop in this paper. We plan to address the job scheduling aspect, more specifically how to shorten the overall map phase completion time of multiple MapReduce jobs, which have different input data block sizes.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

The research presented in this paper is part of the Ph.D. dissertation of the first author under the supervision of the second author. The two authors read and approved the final manuscript.

Authors' information

Wei Dai received the Bachelor of Engineering degree in Computer Engineering in 1999 from Zhejiang University and the MS degree in Computer Engineering in 2009 from the University of Central Florida. He is currently a Ph.D. student in Computer Engineering at the University of Central Florida, Orlando. His research interests include cloud computing, data-intensive computing, and computer networks.
Mostafa Bassiouni received his BS and MS degrees in Computer Science from Alexandria University and received the Ph.D. degree in Computer Science from the Pennsylvania State University in 1982. He is currently a professor of Computer Science at the University of Central Florida, Orlando. His research interests include computer networks, distributed systems, real-time protocols and concurrency control. He has authored and coauthored over 190 papers published in various computer journals, book chapters and conference proceedings. His research has been supported by grants from ARO, ARPA, NSF, STRICOM, PM-TRADE, CBIS, Harris, and the State of Florida. He is an Associate Editor of the Computer Journal- Oxford University Press, Editor-in-Chief of Electronics-MDPI, and an Editorial Board Member of four other journals. He has served as member of the program committee of several conferences, as the program committee chair of CSMA'98 and CSMA'2000 and as the guest co-editor of a special issue of the Journal of Simulation Practice and Theory, 2002.

Received: 1 October 2013 Accepted: 6 December 2013
Published: 18 December 2013

References

1. Gantz JF, Chute C, Manfrediz A, Minton S, Reinsel D, Schlichting W, Toncheva A (2008) The Diverse and Exploding Digital Universe: An updated forecast of worldwide information growth through 2011. IDC White Paper – sponsored by EMC, Framingham, MA, USA
2. Dean J, Ghemawat S (2004) MapReduce: Simplified Data Processing on Large Clusters. Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, Berkeley, CA, USA
3. Official Apache Hadoop Website. http://hadoop.apache.org. Accessed 30 Sep 2013
4. Hadoop Wiki. http://wiki.apache.org/hadoop/PoweredBy. Accessed 30 Sep 2013
5. Wikipedia: Apache Hadoop. http://en.wikipedia.org/wiki/Hadoop. Accessed 30 Sep 2013
6. Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/. Accessed 30 Sep 2013
7. Gottfrid D (2007) Self-service, prorated supercomputing fun. http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/. Accessed 30 Sep 2013
8. White T (2012) Hadoop: The Definitive Guide, 3rd edition. O'Reilly Media, Sebastopol, CA, USA
9. Lin J, Dyer C (2010) Data-Intensive Text Processing with MapReduce. Morgan and Claypool Publishers, San Rafael, CA, USA
10. Zaharia M, Konwinski A, Joseph A, Katz R, Stoica I (2008) Improving MapReduce Performance in Heterogeneous Environments. Proceedings of the 8th USENIX conference on Operating Systems Design and Implementation, Berkeley, CA, USA, pp 29–42
11. Dessouky M, Lageweg B, Lenstra J, van de Velde S (1990) Scheduling identical jobs on uniform parallel machines. Statistica Neerlandica 44:115–123
12. Jiang D, Ooi BC, Shi L, Wu S (2010) The performance of MapReduce: an in-depth study. Proc VLDB Endowment 3(1–2):472–483
13. Lee K, Lee Y, Choi H, Chung YD, Moon B (2011) Parallel data processing with MapReduce: a survey. ACM SIGMOD Rec 40(4):11–20
14. Vijayalakshmi V, Akila A, Nagadivya S (2012) The Survey on MapReduce. Int J Eng Sci Technol 4:07

15. Pinedo M (2012) Scheduling: Theory, Algorithms, and Systems. Springer, New York, NY, USA
16. Lawler E, Lenstra J, Rinnooy Kan A (1982) Recent Developments in Deterministic Sequencing and Scheduling. Deterministic and Stochastic Scheduling. Springer Netherlands, Dordrecht, Netherlands, pp 35–73

doi:10.1186/2192-113X-2-23
Cite this article as: Dai and Bassiouni: An improved task assignment scheme for Hadoop running in the clouds. *Journal of Cloud Computing: Advances, Systems and Applications* 2013 **2**:23.