International Journal of Computer Science Trends and Technology (IJCST) – Volume 5 Issue 2, Mar – Apr 2017

RESEARCH ARTICLE OPEN ACCESS

An Efficient Dynamic Slot Allocation Based On Fairness Consideration for MAPREDUCE Clusters

T. P. Simi Smirthiga [1], P.Sowmiya [2], C.Vimala [3], Mrs P.Anantha Prabha [4]
U.G Scholar [1], [2] & [3], Associate Professor [4]
Department of Computer Science & Engineering
Sri Krishna College of Technology,
Kovaipudur, Coimbatore
Tamil Nadu -India

ABSTRACT
Map Reduce is the popular parallel computing paradigm for large-scale data processing in clusters and data centers. However, the slot utilization can be low, especially when Hadoop Fair Scheduler is used, due to the pre-allocation of slots among reduce tasks and map, and the order that map tasks followed by reduce tasks in a typical MapReduce environment. On permitting slots to be dynamically (re)allocated to either reduce tasks or map depending on their actual requirement this problem is solved. The proposal of two types of Dynamic Hadoop Fair Scheduler (DHFS), for two different levels of fairness (i.e., cluster and pool level) improvise the makespan. HFS is a two-level hierarchy, with task slots allocation across "pools" at the top level, and slots allocation among multiple jobs within the pool at the second level. On proposing two types of Dynamic Hadoop Fair Scheduler (DHFS), with the consideration of different levels of fairness (i.e., pool level and cluster-level). The experimental results show that the proposed DHFS can improve the system performance significantly (by 32% - 55% for a single job and 44% - 68% for multiple jobs) while guaranteeing the fairness.

Keywords:- Mapreduce, Namenode,Jobtracker, Task Tracker, Datanode

I. INTRODUCTION

MapReduce has become an important paradigm for parallel data-intensive cluster programming due to its simplicity and flexibility. Essentially, it is a software framework that allows a cluster of computers to process a large set of unstructured or structured data in parallel. MapReduce users are quickly growing. Apache Hadoop is an open source implementation of MapReduce that has been widely adopted in the industrial area. With the rise of cloud computing, it becomes more convenient for IT business to set a cluster of servers in the cloud and launch a batch of MapReduce jobs. Therefore, by using the MapReduce framework there are a large variety of data-intensive applications. In a classic Hadoop system, each MapReduce job is partitioned into small tasks which are distributed and were executed across multiple machines. There are two kinds of tasks, i.e., map tasks and reduce tasks. Each map task applies the same map function to process a block of the input data and produces a intermediate results in the format of key-value pairs. The intermediate data will be partitioned by hash functions and fetched by the corresponding reduce task as their inputs. Once all the intermediate data has been fetched, a reduce task starts to execute and produce the final results. The Hadoop implementation closely resembles the MapReduce framework. A single master node is adopted to manage the distributed slave nodes. The master node communicates with slave nodes with heartbeat messages which consist of status information of slaves. Job scheduling is performed by a centralized job tracker routine in the master node. The scheduler assigns tasks to slave nodes which have free resources and response to heartbeats as well. The resources in each slave node are represented as map/reduce slots. Each slave node has a fixed number of slots, and each map slot only processes one map task at a moment.

II. LITERATURE REVIEW

Bi-criteria algorithm for a scheduling job uses a new method for building an efficient algorithm for scheduling jobs in a cluster. Here jobs are considered as parallel tasks (PT) which can be scheduled on any number of processors. The main thing is to consider two criteria that are optimized together. These criteria are the weighted minimal average completion time (minsum) and the makespan. They are chosen for their complementarity, to be able to represent both system administrator objectives and user-oriented objectives. In this paper, we presented a new algorithm for scheduling a set of independent jobs on a cluster. The main feature is to optimize two criteria simultaneously. The experiments show that in average the performance ratio is very good, and the algorithm is fast enough for practical use.[1]

An adoptive disk I/O scheduling on virtualized environment is challenging problem for performance predictability and system throughput for large-scale virtualized environments in the cloud. Virtualization has become the fundamental technique for cloud computing. As data-intensive applications become popular in the cloud, their performance on the virtualized platform calls for empirical evaluations and technical innovations. In this study,by investigating the performance of the popular data-intensive system, MapReduce, on a virtualized platform. The detailed study reveals that different disk pair schedulers within the virtual machine manager and virtual machines cause a noticeable variation in the Hadoop performance in virtual cluster. Address this problem through developing a meta-scheduler for selecting a suitable disk pair schedulers. Given an application, a program is divided into phases; currently coarse-grained phase detection is used, using the program progress. A novel heuristic successively evaluates solutions (phase-scheduler assignments). The program is executed, and the performance score is measured. Then, the heuristic evaluates the solution based on the best performance score. Performance results are obtained on our local virtual cluster which is based on a Xen hypervisor. It is found that for most of the Hadoop benchmarks, using multiple pairs in a single application can provide a better performance score over any single-pair solution. More importantly, our solution provides up to 25% performance improvement over the default virtual Hadoop cluster configuration. Moreover, the performance improvement is proportional to the data size, VM consolidation and system scale.[4]

The map-reduce paradigm is now standard in academia and industry for processing large-scale data. In this work, formalize job scheduling in map-reduce as a novel generalization of the two-stage classical flexible flow shop (FFS) problems: instead of a single task at each stage, a job now consists of a set of tasks per stage. For this generalization, consider the problem of minimizing the total flowtime ,given an efficient 12-approximation in the offline setting and an online competitive algorithm. Motivated by map-reduce, revisit the two-stage flow shop problem, where we give a dynamic program for minimizing the total flowtime when all the jobs arrive at the same time. To provide a simple formalization of the scheduling problem in the map-reduce framework, many issues have been ignored in real systems that often have a large effect on the performance. On assuming that the scheduler is aware of the job sizes. These may not be immediately available in practice, but in nearly all circumstances approximate job sizes can be determined based on historical facts the bottle neck scenario is avoided.[6]

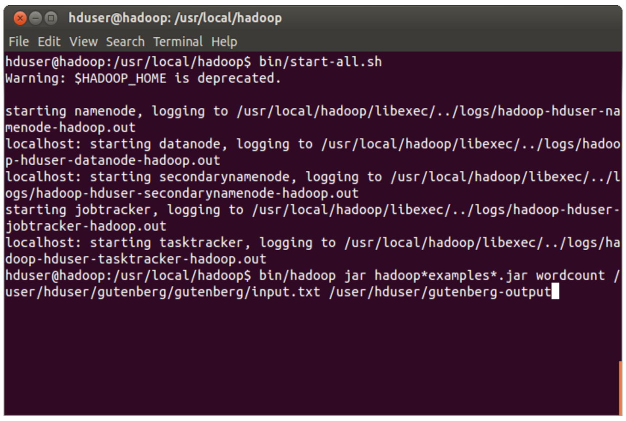
III. PROPOSED METHODOLOGY

A. MapReduce Programming Model

MapReduce is the popular programming model for processing large data sets, initially proposed by Google. Now it has been a de facto standard for large scale data processing on the cloud. Hadoop is an open-source java implementation of MapReduce. When a user submits jobs to the Hadoop cluster, Hadoop system breaks each job into multiple map tasks and reduce tasks. Each map task processes (i.e. records and scans) a data block and produces intermediate results in the form of key-value pairs. Generally, the number of map tasks for a job is determined by the input data. There is one map task per data block. The execution time for a map task is determined by the data size of an input block. The reduce tasks consists of shuffle/sort/reduce phases. In the shuffle phase, the reduce task fetch the intermediate outputs from each map task. In the sort/reduce phase, the reduce tasks sort intermediate data and then aggregate the intermediate values for each key to produce the final output. The number of reduce tasks for each job is not determined, which depends on the intermediate map outputs.

MapReduce suffers from a under-utilization of the respective slots as the number of map and reduce tasks varies over time, resulting in occasions where the number of slots allocated for map/reduce is smaller than the number of map/reduce tasks. Our dynamic slot allocation policy is based on the observation that at different period of time there may be idle map (or reduce) slots. The unused map slots for those overloaded reduce tasks to improve the performance of the MapReduce workload. For example, at the beginning of MapReduce workload computation, there will be no computing reduce tasks and only computing map tasks, i.e., all the computation workload lies in the map-side. In that case, make use of idle reduce slots for running map tasks. That is, we break the implicit assumption for current MapReduce framework that the map tasks can only run on map slots and reduce tasks can only run on reduce slots. Instead, we modify it as follows: both map and reduce tasks can be run on either map or reduce slots.

- When the user submits jobs to the Hadoop cluster, Hadoop system breaks each job into multiple map tasks and reduce tasks.
- Each map task processes (i.e. scans and records) a data block and produces intermediate results in the form of key-value pairs.
- The execution time for a map task is determined by the data size of an input block.
- The reduce tasks consists of shuffle/sort/reduce phases. In the shuffle phase, the reduce task fetch the intermediate outputs from each map task.
- In the sort/reduce phase, the reduce tasks sort intermediate data and then aggregate the intermediate values for each key to produce the final output.



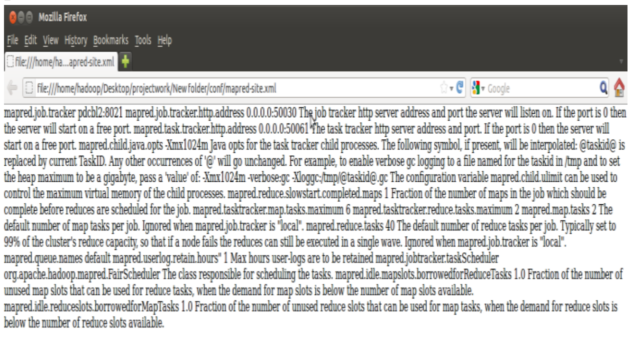
```
hduser@hadoop:/usr/local/hadoop$ bin/start-all.sh
Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-na
menode-hadoop.out
localhost: starting datanode, logging to /usr/local/hadoop/libexec/../logs/hadoo
p-hduser-datanode-hadoop.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop/libexec/../l
ogs/hadoop-hduser-secondarynamenode-hadoop.out
starting jobtracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-
jobtracker-hadoop.out
localhost: starting tasktracker, logging to /usr/local/hadoop/libexec/../logs/ha
doop-hduser-tasktracker-hadoop.out
hduser@hadoop:/usr/local/hadoop$ bin/hadoop jar hadoop*examples*.jar wordcount /
user/hduser/gutenberg/gutenberg/input.txt /user/hduser/gutenberg-output
```

hduser@hadoop: /usr/local/hadoop
File Edit View Search Terminal Help

Fig.1 MapReduce Installation

B. Pool-Independent Fairness Scheduler

In contrast to DHFS that considers the fairness in its dynamic slots allocation independent of pools, but rather across typed-phases, there is another alternative fairness consideration for the dynamic slots allocation across pools, as we call Pool-dependent DHFS DHFS). It assumes that each pool, consisting of two parts: map-phase pool and reduce-phase pool, is selfish. That is, it always tries to satisfy its own shared map and reduce slots for its own needs at the map-phase and reduce-phase as much as possible before lending them to other pools DHFS will be done with the following two processes:
(1). Intra-Pool dynamic slots allocation. First, each typed phase pool will receive its share of typed-slots based on max-min fairness at each phase. There are four possible relationships for each pool regarding its demand (denoted as map Slots Demand, reduce Slots Demand) and its share (marked as map Share, reduce Share) between two phases.



Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///home/ha...apred-site.xml
file:///home/hadoop/Desktop/projectwork/New folder/conf/mapred-site.xml
Google

```
mapred.job.tracker pdcbl2:8021 mapred.job.tracker.http.address 0.0.0.0:50030 The job tracker http server address and port the server will listen on. If the port is 0 then the server will start on a free port. mapred.task.tracker.http.address 0.0.0.0:50061 The task tracker http server address and port. If the port is 0 then the server will start on a free port. mapred.child.java.opts -Xmx1024m Java opts for the task tracker child processes. The following symbol, if present, will be interpolated: @taskid@ is replaced by current TaskID. Any other occurrences of '@' will go unchanged. For example, to enable verbose gc logging to a file named for the taskid in /tmp and to set the heap maximum to be a gigabyte, pass a 'value' of: -Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc The configuration variable mapred.child.ulimit can be used to control the maximum virtual memory of the child processes. mapred.reduce.slowstart.completed.maps 1 Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job. mapred.tasktracker.map.tasks.maximum 6 mapred.tasktracker.reduce.tasks.maximum 2 mapred.map.tasks 2 The default number of map tasks per job. Ignored when mapred.job.tracker is "local". mapred.reduce.tasks 40 The default number of reduce tasks per job. Typically set to 99% of the cluster's reduce capacity, so that if a node fails the reduces can still be executed in a single wave. Ignored when mapred.job.tracker is "local".
mapred.queue.names default mapred.userlog.retain.hours" 1 Max hours user-logs are to be retained mapred.jobtracker.taskScheduler
org.apache.hadoop.mapred.FairScheduler The class responsible for scheduling the tasks. mapred.idle.mapslots.borrowedforReduceTasks 1.0 Fraction of the number of unused map slots that can be used for reduce tasks, when the demand for map slots is below the number of map slots available.
mapred.idle.reduceslots.borrowedforMapTasks 1.0 Fraction of the number of unused reduce slots that can be used for map tasks, when the demand for reduce slots is below the number of reduce slots available.
```

Fig.2. Hadoop Fairness Schedular File

C. Job Execution

Word Count example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab. Each mapper take a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the sum and word. As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record.

IV. RESULT

The proposed system improves the utilization and performance for MapReduce clusters while guaranteeing the fairness across pools. PI-DHFS and PD-DHFS are our first two attempts of achieving this goal with two different fairness definitions. PI-DHFS follows strictly the definition of fairness given by traditional HFS, i.e., the slots are fairly shared across pools within each phase (i.e., map phase or reduce phase). However, the slot allocations are independent across phases. In contrast, PD-DHFS gives a new definition of fairness from the perspective of pools, i.e., each pool shares the total number of map and reduce slots from the map phase and reduce phase fairly with other pools.

V. PROCESS

A. Classify the Slots

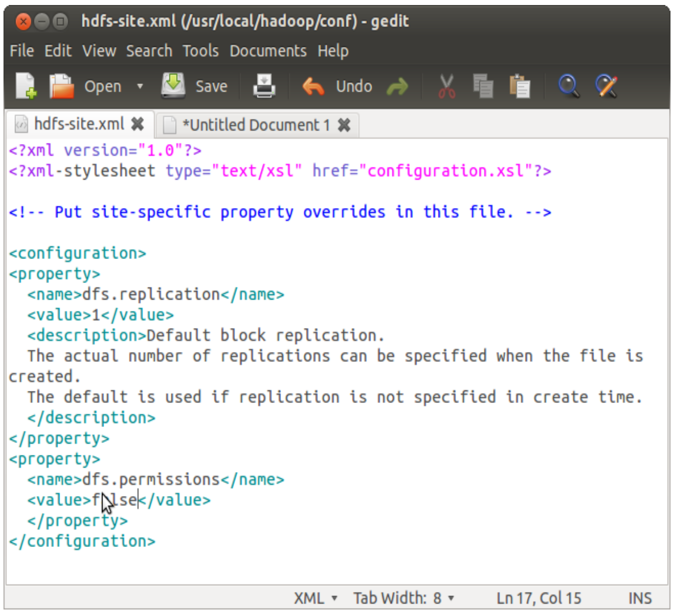
The performance of a MapReduce cluster via optimizing the slots utilization primarily from two perspectives. First, classify the slots into two types, namely, idle slots (i.e., no running tasks) and busy slots (i.e., with running tasks) . Given the total number of map and reduce slots configured by users, one optimization approach (i.e., macro-level optimization) is to improve the slot utilization by maximizing the number of busy slots and reducing the number of idle slots. Second, it is worth noting that not every busy slot can be efficiently utilized. Thus, our optimization approach (i.e., micro-level optimization) is to improve the utilization efficiency of busy slots after the macrolevel optimization. Particularly, we identify two main affecting factors: (1). Speculative tasks; (2). Data locality. Based on these, we propose Dynamic a dynamic utilization optimization framework for MapReduce, to improve the performance of a shared Hadoop cluster under a fair scheduling between users.

B. Dynamic Hadoop Slot Allocation

Current design of MapReduce suffers from a under-utilization of the respective slots as the number of map and reduce tasks varies over time, resulting in occasions where the number of slots allocated for map/reduce is smaller than the number of map/reduce tasks. Our dynamic slot allocation policy is based on the observation at different period of time there may be idle map (reduce) slots, as the job proceeds from map phase to reduce phase. We can use the unused map slots are those overloaded reduce tasks to improve the performance of the MapReduce workload, and vice versa. For example, at the beginning of MapReduce workload computation, there will be no computing reduce tasks and only computing map tasks, i.e., all the computation the density-based clustering corresponding to a broad range of parameter settings.

C. Fairness Calculation

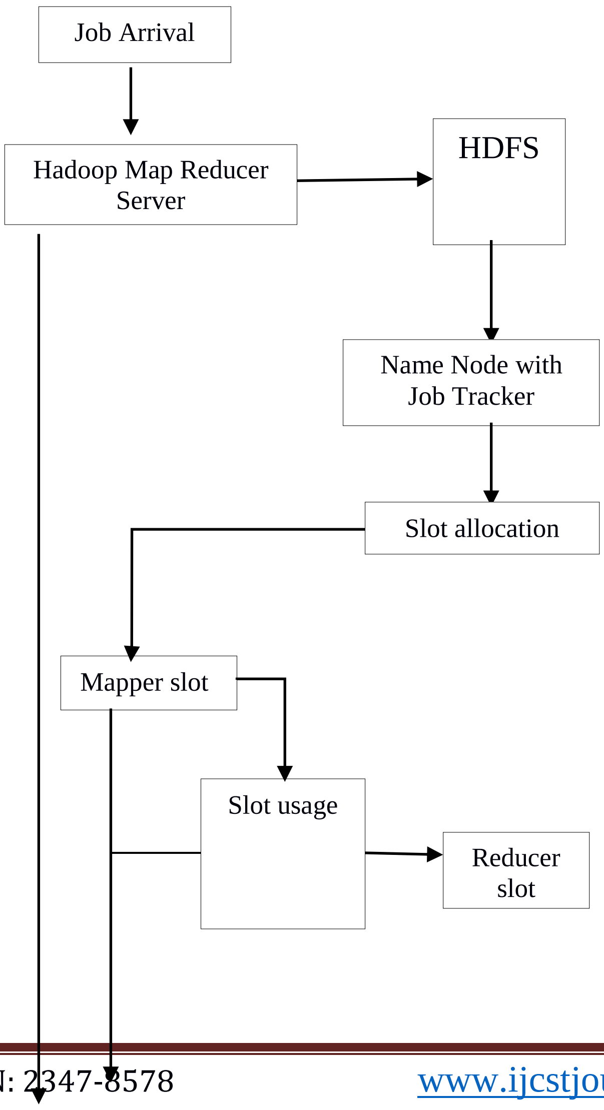
Fairness is an important metric in Hadoop Fair Scheduler. It is fair when all pools have been allocated with the same amount of resources. In HFS, task slots are first allocated across the pools, and then the slots are allocated to the jobs within the pool. Moreover, a MapReduce job computation consists of two parts: map-phase task computation and reduce-phase task computation. One question is about how to define and ensure fairness under the dynamic slot allocation policy. The resource requirements between the map slot and reduce slot are generally different. This is because the map task and reduce task often exhibit completely different execution patterns. Reduce task tends to consume much more resources such as memory and network bandwidth. Simply allowing reduce tasks to use map slots requires configuring each map slot to take more resources, which will consequently reduce the effective number of slots on each node, causing resource under-utilized during runtime. Thus, a careful design of dynamic allocation policy is important and needed to be aware of such difference.

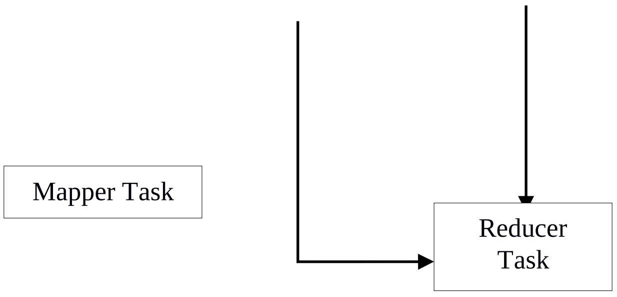

hdfs-site.xml (/usr/local/hadoop/conf) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
hdfs-site.xml
*Untitled Document 1
XML
Tab Width: 8
Ln 17, Col 15
INS

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is
created.
  The default is used if replication is not specified in create time.
  </description>
</property>
<property>
  <name>dfs.permissions</name>
  <value>false</value>
  </property>
</configuration>
```

Fig.3. HDFS File Properties Specification

FLOW CHART





VI. CONCLUSION

The system developed an efficient fairness scheduler by implementing pool distribution of job arrival around the clusters. The future step of our research includes integration of more advance scheduling management to include resource discovery means for demonstrating the effectiveness of our deadline algorithm in dynamic node formation. In addition, we aim of applying the solution into large-scale virtualized grids and inter-cloud scenarios to explore the efficiency of the algorithm in highly dynamic and large-scale cases.

REFERENCES

[1] P.-F. Dutot, L. Eyraud, G. Mounie, and D. Trystram, “Bi-criteria algorithm for scheduling jobs on cluster platforms,” in Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit., 2004, pp. 125–132.

[2] P.-F.Dutot, G.Mounie, and D. Trystram,“Scheduling parallel tasks: Approximation algorithms,” in Handbo ok of Scheduling: Algorithms, Models, and Performance Analysis, J. T. Leung, d. Boca Raton, FL, USA: CRC Press, ch. 26, pp. 26-1–26-24.

[3] J. Gupta, A. Hariri, and C. Potts, “Scheduling a two-stage hybrid flow shop with parallel machines at the first stage,” Ann. Oper. Res., vol. 69, pp. 171–191, 1997.

[4] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, “Adaptive disk I/O scheduling for mapreduce in virtualized environment,” in Proc. Int. Conf. Parallel Process., Sep. 2011, pp. 335–344.

[5] G. J. Kyparisis and C. Koulamas, "A note on makespan minimization in two-stage flexible flow shops with uniform machines," Eur. J. Oper. Res., vol. 175, no. 2, pp. 1321–1327, 2006.

[6] B. Moseley, A. Dasgupta, R. Kumar, and T.Sarlos, "On scheduling in map-reduce and flow-shops," in Proc. 23rd Annu. ACM Symp. Parallelism Algorithms Archit., 2011, pp. 289–298.

[7] C. Ouz, M. F. Ercan, T. E. Cheng, and Y. Fung, "Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop," Eur. J. Oper. Res., vol. 149, no. 2, pp. 390–403, 2003.

[8] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres,and E. Ayguade, "Deadline-based mapreduce workload management," IEEE Trans. Netw. Service Manage., vol. 10, no. 2, pp. 231–244, Jun. 2013.

[9] P. Sanders and J. Speck, "Efficient parallel scheduling of malleable tasks," in Proc. IEEE Int. Parallel Distrib. Process. Symp., 2011, pp. 1156–1166.

[10] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-55, Apr. 2009.