Improving Performance of Heterogeneous MapReduce Clusters with Adaptive Task Tuning

Dazhao Cheng, *Member, IEEE*, Jia Rao, *Member, IEEE*, Yanfei Guo, *Member, IEEE*, Changjun Jiang, *Member, IEEE*, and Xiaobo Zhou, *Senior Member, IEEE*

Abstract—Datacenter-scale clusters are evolving toward heterogeneous hardware architectures due to continuous server replacement. Meanwhile, datacenters are commonly shared by many users for quite different uses. It often exhibits significant performance heterogeneity due to multi-tenant interferences. The deployment of MapReduce on such heterogeneous clusters presents significant challenges in achieving good application performance compared to in-house dedicated clusters. As most MapReduce implementations are originally designed for homogeneous environments, heterogeneity can cause significant performance deterioration in job execution despite existing optimizations on task scheduling and load balancing. In this paper, we observe that the homogeneous configuration of tasks on heterogeneous nodes can be an important source of load imbalance and thus cause poor performance. Tasks should be customized with different configurations to match the capabilities of heterogeneous nodes. To this end, we propose a self-adaptive task tuning approach, *Ant*, that automatically searches the optimal configurations for individual tasks running on different nodes. In a heterogeneous cluster, Ant first divides nodes into a number of homogeneous subclusters based on their hardware configurations. It then treats each subcluster as a homogeneous cluster and independently applies the self-tuning algorithm to them. Ant finally configures tasks with randomly selected configurations and gradually improves tasks configurations by reproducing the configurations from best performing tasks and discarding poor performing configurations. To accelerate task tuning and avoid trapping in local optimum, Ant uses genetic algorithm during adaptive task configuration. Experimental results on a heterogeneous physical cluster with varying hardware capabilities show that Ant improves the average job completion time by 31, 20, and 14 percent compared to stock Hadoop (Stock), customized Hadoop with industry recommendations (Heuristic), and a profiling-based configuration approach (Starfish), respectively. Furthermore, we extend Ant to virtual MapReduce clusters in a multi-tenant private cloud. Specifically, Ant characterizes a virtual node based on two measured performance statistics: I/O rate and CPU steal time. It uses k-means clustering algorithm to classify virtual nodes into configuration groups based on the measured dynamic interference. Experimental results on virtual clusters with varying interferences show that Ant improves the average job completion time by 20, 15, and 11 percent compared to Stock, Heuristic and Starfish, respectively.

Index Terms—MapReduce performance improvement, self-adaptive task tuning, heterogeneous clusters, genetic algorithm

1 Introduction

In the past few years, MapReduce has proven to be an effective platform to process a large set of unstructured data as diverse as sifting through system logs, running extract transform load operations, and computing web indexes. Since big data analytics requires distributed computing at scale, usually involving hundreds to thousands of machines, access to such facilities becomes a significant barrier to practicing big data processing in small business. Deploying MapReduce in datacenters or cloud platforms, offers a more cost-effective model to implement big data analytics. However, the heterogeneity in datacenters and clouds present significant challenges in achieving good MapReduce performance [1], [2].

Hardware heterogeneity occurs because servers are gradually upgraded and replaced in datacenters. Interferences from multiple tenants sharing the same cloud platform can also cause heterogeneous performance even on homogeneous hardware. The difference in processing capabilities on MapReduce nodes breaks the assumption of homogeneous clusters in MapReduce design and can result in load imbalance, which may cause poor performance and low cluster utilization. To improve MapReduce performance in heterogeneous environments, work has been done to make task scheduling [2], [3] and load balancing [1], [4] heterogeneity aware. Despite these optimizations, most MapReduce implementations such as Hadoop still perform poorly in heterogeneous environments. For the ease of management, MapReduce implementations use the same configuration for tasks. Existing research [5], [6] has shown that MapReduce configurations should be set according to cluster size and hardware configurations. Thus, running tasks with homogeneous configurations on heterogeneous nodes inevitably leads to sub-optimal performance.

- D. Cheng is with the Department of Computer Science, University of North Carolina at Charlotte, NC 28223. E-mail: dazhao.cheng@uncc.edu.
- J. Rao and X. Zhou are with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918. E-mail: {jrao, xzhou}@uccs.edu.
- C. Jiang is with the Department of Computer Science & Technology, Tongji University, 4800 Caoan Road, Jiading, Shanghai 201804, China. E-mail: cjjiang@tongji.edu.cn.
- Y. Guo is currently a Postdoc Fellow in the Argonne National Lab, Lemont, IL 60439. E-mail: yguo@anl.gov.

Manuscript received 14 Oct. 2015; revised 14 June 2016; accepted 12 July 2016. Date of publication 27 July 2016; date of current version 15 Feb. 2017.
Recommended for acceptance by F. Cappello.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2594765

1045-9219 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

In this work, we propose a task tuning approach that allows tasks to have different configurations, each optimized for the actual hardware capabilities, on heterogeneous nodes. We address the following challenges in automatic MapReduce task tuning. First, determining the optimal task configuration is a tedious and error-prone process. A large number of performance-critical parameter can have complex interplays on task execution. Previous studies [6], [7], [8], [9] have shown that it is difficult to construct models to connect parameter settings with MapReduce performance. Second, there is no one-size-fit-all model for different MapReduce jobs, and even different configurations are needed for different execution phases or input sizes. In a cloud environment, task configurations should also be changed in response to the changes in multi-tenancy interferences. Finally, most MapReduce implementations use fixed task configurations that are set during job initializations [10]. Adaptive task tuning requires new mechanisms for on-the-fly task reconfiguration.

We present *Ant*, a self-adaptive tuning approach for task configuration in heterogeneous environments. Ant monitors task execution in large MapReduce jobs, which comprise multiple waves of tasks and optimizes task configurations as job execution progresses. It clusters worker nodes (either physical or virtual nodes) into groups according to their hardware configurations or the estimated capability based on interference statistics. For each node group, Ant launches tasks with different configurations and considers the ones that complete sooner as good settings. To accelerate tuning speed and avoid trapping in local optimum, Ant uses genetic functions *crossover* and *mutation* to generate task configurations for the next wave from the two best performing tasks in a group. We implement Ant in Hadoop, the popular open source implementation of MapReduce, and perform comprehensive evaluations with representative MapReduce benchmark applications. Experimental results on a physical cluster with three types of machines show that Ant improves the average job completion time (JCT) by 31, 20, and 14 percent compared to stock Hadoop (Stock), customized Hadoop with industry recommendations (Heuristic), and a profiling-based configuration approach (Starfish), respectively. Our results also show that although Ant is not quite effective for small jobs with only a few waves of tasks, it can significantly improve the performance of large jobs. Experiments with Microsoft's MapReduce workload, which consist of 10 percent large jobs, demonstrate that Ant is able to reduce the overall workload completion time by 12.5 and 8 percent compared to heuristic- and profiling-based approaches.

A preliminary version of the paper appeared in [11]. In this manuscript, we have extended Ant to virtual MapReduce clusters in multi-tenancy private cloud environments. Specifically, Ant characterizes a virtual node based on two measured performance statistics: I/O rate and CPU steal time. We consider two representative interference scenarios in the cloud: stable interference and dynamic interference. Experimental results on virtual clusters with varying interferences show that Ant improves the average job completion time by 20, 15, and 11 percent compared to Stock, Heuristic and Starfish, respectively.

The rest of this paper is organized as follows. Section 2 gives motivations on improvement of MapReduce configuration framework. Section 3 describes the design of Ant.

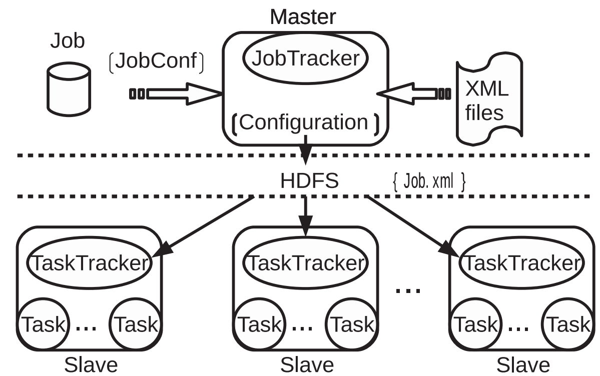


Fig. 1. The Hadoop framework.

Section 4 presents the details of the proposed self-tuning algorithm. Section 5 introduces moving Ant into the cloud. Section 6 gives Ant implementation details. Section 7 presents the experimental results and analysis on a physical cluster. Section 8 presents the experimental results and analysis on a virtual cluster. Section 9 reviews related work. Section 10 concludes the paper.

2 Motivations

2.1 Background

MapReduce is a distributed parallel programming model originally designed for processing a large volume of data in a homogeneous environment. Based on the default Hadoop framework, a large number of parameters need to be set before a job can run in the cluster. These parameters control the behaviors of jobs during execution, including their memory allocation, level of concurrency, I/O optimization, and the usage of network bandwidth. As shown in Fig. 1, slave nodes load configurations from the master node where the parameters are configured manually. By design, tasks belonging to the same job share the same configuration.

In Hadoop, there are more than 190 configuration parameters, which determine the settings of the Hadoop cluster, describe a MapReduce job to the Hadoop framework, and optimize task execution [10]. Cluster-level parameters specify the organization of a Hadoop cluster and some long-term static settings. Changes to such parameters require rebooting the cluster to take effect. Job-level parameters determine the overall execution settings, such as input format, number of map/reduce tasks, and failure handling. These parameters are relatively easier to tune and have uniform effect on all tasks even in a heterogeneous environment. Task-level parameters control the fine-grained task execution on individual nodes and can possibly be changed independently and on-the-fly at runtime. Parameter tuning at the task level opens up opportunities for improving performance in heterogeneous environments and is our focus in this work.

Hadoop installations pre-set the configuration parameters to default values assuming a reasonably sized cluster and typical MapReduce jobs. These parameters should be specifically tuned for a target cluster and individual jobs to achieve the best performance. However, there is very limited information on how the optimal settings can be determined. There exist rule of thumb recommendations from industry leaders (e.g., Cloudera [12] and MapR [13]) as well as academic studies [6], [8]. These approaches can not be

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Multiple Machine Types in the Cluster

Machine model	CPU	Memory	Disk
Supermicro Atom	4*2.0 GHz	8 GB	1 TB
PowerEdge T110	8*3.2 GHz	16 GB	1 TB
PowerEdge T420	24*1.9 GHz	32 GB	1 TB

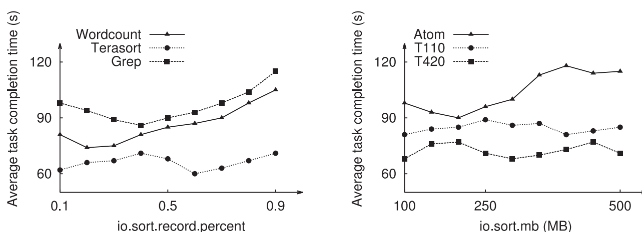
universally applied to a wide range of applications or heterogeneous environments. In this work, we develop an online self-tuning approach for task-level configuration. Next, we provide motivating examples to show the necessity of configuration tuning for heterogeneous workloads and hardware platforms.

2.2 Motivating Examples

We created a heterogeneous Hadoop cluster composed of three types of machines listed in Table 1. Three MapReduce applications from the PUMA benchmark [14], i.e., *WordCount*, *Terasort* and *Grep*, each with 300 GB input data, were run on the cluster. We configured each slave node with four map slots and two reduce slots, and HDFS block size was set to 256 MB. The heap size `mapred.child.java.opts` was set to 1 GB and other parameters were set to the default values. We measured the map task completion time in two different scenarios—heterogeneous workload on homogeneous hardware and homogeneous workload on heterogeneous hardware. We show that the heterogeneity either in the workload or hardware makes the determination of the optimal task configuration difficult.

Fig. 2a shows the average map completion times of the three heterogeneous benchmarks on a homogeneous cluster only consisting of the T110 machines. The completion times changed as we altered the values of parameter `io.sort.record.percent`. The figure shows that *wordcount*, *terasort*, and *grep* achieved their minimum completion times when the parameter was set to 0.4, 0.2, and 0.6, respectively. Fig. 2b shows the performance of *wordcount* on machines with different hardware configurations. Map completion times varied as we changed the value of parameter `io.sort.mb`. The figure suggests that uniform task configurations do not lead to the optimal performance in a heterogeneous environment. For example, map tasks achieved the best performance on the Atom machine when the parameter was set to 125 while the optimal completion time on the T420 machine was due to the parameter being set to 275.

Summary. We have shown that the performance of Hadoop applications can be substantially improved by tuning task-level parameters for heterogeneous workloads and platforms.



(a) Heterogeneous workloads. (b) Heterogeneous platforms.

Fig. 2. The optimal task configuration changes with workloads and platforms.

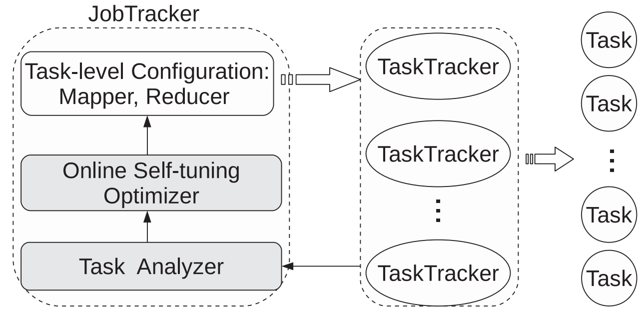


Fig. 3. The architecture of Ant.

However, parameter optimization is an error-prone process involving complex interplays among the job, the Hadoop framework and the architecture of the cluster. Furthermore, manual tuning still remains difficult due to the large parameter search space. As many MapReduce jobs are recurring or have multiple waves of task execution, it is possible to learn the best task configurations based on the feedback of previous runs. These observations motivated us to develop a task self-tuning approach to automatically configure parameters for various Hadoop jobs and platforms in an online manner.

3 Ant Design and Assumptions

3.1 Architecture

Ant is a self-tuning approach for multi-wave MapReduce applications, in which job executions consist of several rounds of map and reduce tasks. Unlike traditional MapReduce implementations, Ant centers on two key designs: (1) tasks belonging to the same job run with different configurations matching the capabilities of the hosting machines; (2) the configurations of individual tasks dynamically change to search for the optimal settings. Ant first spawns tasks with random configurations and executes them in parallel. Upon task completion, Ant collects task runtimes and adaptively adjusts task settings according to the best-performing tasks. After several rounds of tuning, task configurations on different nodes converge to the optimal settings. Since task tuning starts with random settings and improves with job execution, ant does not require any priori knowledge of MapReduce jobs and is model independent. Fig. 3 shows the architecture of Ant.

- *Self-tuning optimizer* uses a genetic algorithm (GA)-based approach to generate task configurations based on the feedback reported by the *task analyzer*. Settings that are top-ranked by the task analyzer are used to re-produce the optimized configurations.
- *Task analyzer* uses a fitness (utility) function to evaluate the performance of individual tasks due to different configurations. The fitness function takes into account task completion time as well as other performance critical execution statistics.

Ant operates as follows. When a job is submitted to the `JobTracker`, the configuration optimizer generates a set of parameters randomly in a reasonable range to initialize the task-level configuration. Then the `JobTracker` sends the randomly initialized tasks to their respective `TaskTrackers`. The steps of task tuning correspond to the multiple waves of tasks execution. Upon completing a wave, the task analyzer residing in the `JobTracker` recommends good configurations to the configuration

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

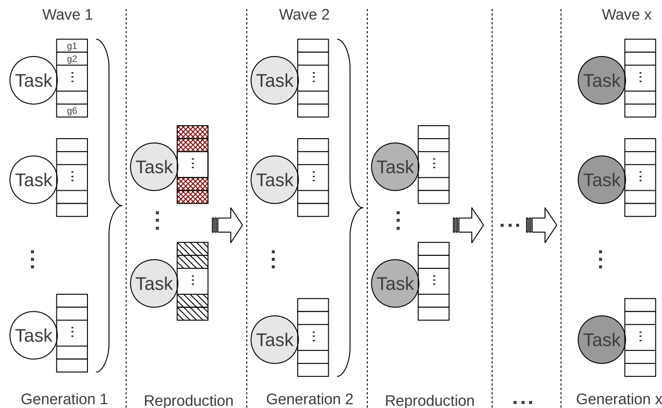


Fig. 4. Task self-tuning process in Ant.

optimizer for the next wave of execution. This process is repeated until the job completes.

3.2 Assumptions

Our findings that uniform task configurations lead to sub-optimal performance in a heterogeneous environment motivated the design of Ant, a self-tuning approach that allows differentiated task settings in the same job. The effectiveness of Ant relies on two assumptions—substantial performance improvement can be achieved via task configurations and the MapReduce jobs are long running ones (e.g., with multiple waves) which allow for task reconfiguration and performance optimization. There are two levels of heterogeneity that can affect task performance, i.e., task-level data skewness and machine-level varying capabilities. Although due to data skew some tasks inherently take longer to finish, Ant assumes that the majority of tasks have uniform completion time with identical configurations. Ant focuses on improving performance for average tasks by matching task configurations to the actual hardware capabilities. To address hardware heterogeneity, Ant groups nodes with similar hardware configurations or capabilities together and compares parallel executing tasks to determine the optimal configurations for the node group. However, task skew and varying hardware capabilities due to interferences in multi-tenant clouds can possibly impede getting good task configurations.

4 SELF-ADAPTIVE TUNING

Ant identifies good configurations by comparing the performance of parallel executing tasks on the nodes with similar processing capabilities in a self-tuning manner. Due to the multi-wave task execution in many MapReduce jobs, Ant is able to continuously improve performance by adapting task configurations. In this section, we first describe how Ant forms self-tuning task groups in which different configurations can be compared. We then discuss the set of parameters Ant optimizes and the utility function Ant uses to evaluate the goodness of parameters. Finally, we present the design of a genetic algorithm-based self-tuning approach and a strategy to accelerate the tuning speed.

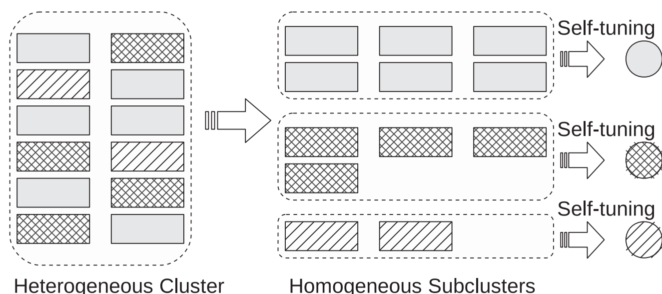


Fig. 5. Ant on a heterogeneous cluster.

TABLE 2
Task-Level Parameters and Search Space

Task-level parameters	Search space	Symbol
io.sort.factor	{1, 300}	g_1
io.sort.mb	{100, 500}	g_2
io.sort.record.percent	{0.05, 0.8}	g_3
io.sort.spill.percent	{0.1, 0.9}	g_4
io.file.buffer.size	{4 K, 64 K}	g_5
mapred.child.java.opts	{200, 500}	g_6

4.1 Forming Self-Tuning Groups

We begin with describing Ant's workflow in a homogeneous cluster and discuss how to form homogeneous subclusters in a heterogeneous cluster.

Homogeneous Cluster. In a homogeneous cluster, all nodes have the same processing capability. Thus, Ant considers the whole Hadoop cluster as a self-tuning group. Each node in the Hadoop cluster is configured with a predefined number of map and reduce slots. If the number of tasks (e.g., mappers) exceeds the available slots in the cluster (e.g., map slots), execution proceeds in multiple waves. Fig. 4 shows the multi-wave task self-tuning process in a homogeneous cluster. Ant starts multiple tasks with different configurations concurrently in the self-tuning group and reproduces new configurations based on the feedback of completed tasks. We frame the tuning process as an evolution of configurations, in which each wave of execution refers to one configuration generation. The reproduction of generations is directed by a genetic algorithm which ensures that the good configurations in prior generations are preserved in new generations.

Heterogeneous Cluster. In a heterogeneous cluster, as shown in Fig. 5, Ant divides nodes into a number of homogeneous subclusters based on their hardware configurations. Hardware information can be collected by the `JobTracker` on the master node using the heartbeat connection. Ant treats each subcluster as an homogeneous cluster and independently applies the self-tuning algorithm to them. The outcomes of the self-tuning process are significantly improved task-level configurations, one for each subcluster. Since each subcluster has different processing capability, the optimized task configurations can be quite different across subclusters.

4.2 Task-Level Parameters

Task-level parameters control the behavior of task execution, which is critical to the Hadoop. Previous studies have shown that a small set of parameters are critical to Hadoop performance. Thus, as shown in Table 2, we choose task-level parameters which have significant performance impact as the candidates for tuning. We further shrink the initial searching space of these parameters to a reasonable

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

range in order to accelerate the search speed. This simple approach allows us to cut the search time down from a few hours to a few minutes.

4.3 Evaluating Task Configurations

To compare the performance of different task configurations, Ant requires a quantitative metric to rank configurations. As the goal of task tuning is to minimize job execution time, task completion time (TCT) is an intuitive metric to evaluate performance. However, TCT itself is not a reliable metric to evaluate task configurations. A longer task completion time does not necessarily indicate a worse configuration as some tasks are inherently longer to complete. For example, due to data skew, tasks that have expensive records in their input files can take more than five times longer to complete. Thus, we combine TCT with another performance metric to construct a utility function (or a fitness function in genetic algorithms). We found that most task mis-configurations are related to task memory allocations and incur excessive data spill operations. If either of the `kvbuffer` or the metadata buffer fills up, a map task spills intermediate data to local disks. The spills could lead to three times more I/O operations [12], [15]. Thus, Ant is designed to simultaneously minimize task completion time and the number of spills.

We define the fitness function of a configuration candidate (C_i) as: $f(C_i) = \frac{1}{TCT^2(C_i) \times (\#spills)}$, where TCT is the task completion time and $\#spills$ is the number of spill operations. Since majority of tasks have little or no data skew, we give more weight to TCT in the formulation of the fitness function. Task configurations with high fitness values will be favored in the tuning process. Note that the fitness function does not address the issue of data skew due to non-uniform record distributions in task inputs. We believe that configurations optimized for a task with inherently more data can be even harmful to normal tasks as allocating more memory to normal tasks incurs resource waste.

4.4 Task Self-Tuning

Ant deploys an online self-tuning approach based on genetic algorithm to search the optimal task-level configurations. We consider MapReduce jobs composed of multiple waves of map tasks. The performance of individual task T is determined by its parameter set C. A set candidate C_i consisting of a number of selected parameters (refer to genes in GA), denoted as $C_i = [g_1, g_2, \ldots, g_n]$, represents a task configuration set, where n is the number of parameters. Each element g represents a task-level parameter as shown in Table 2.

Reproduction Process. Ant begins with an initial configuration of randomly generated candidates for the task assignment. After that, it evolves individual task configuration to breed good solutions during each interval by using the genetic reproduction operations. As shown in Algorithm 1, Ant first evaluates the fitness of all completed tasks in the last control interval. Note that M represents the total number of the completed tasks in the last control interval. When there is any available slot in the cluster, it selects two configuration candidates as the evolving parents. Ant generates the new generation configuration candidates by using the proposed genetic reproduction operations. Finally, it assigns the task with the new generated configuration set to the available slot.

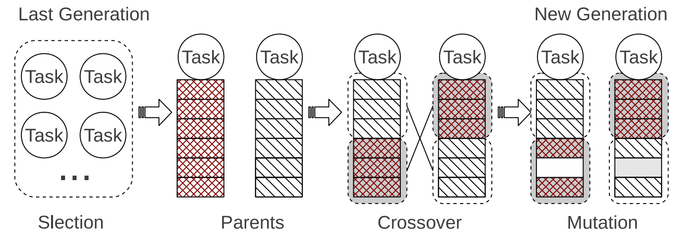


Fig. 6. Reproduction operations.

Algorithm 1. Ant Task Self-Tuning Algorithm

1: /*Evaluate the fitness of each completed task*/
2: $f(C_1), \ldots, f(C_i), \ldots, f(C_M)$
3: **repeat**
4: **if** Any slot is available **then**
5: Select two configuration candidates as parents;
6: Do Crossover and Mutation operations;
7: Use the obtained new generation C_{new} to assign the task to the available slot;
8: **end if**
9: **until** the running job completed.

There are many variations of the reproduction algorithm obtained by altering the *selection*, *crossover*, and *mutation* operators as shown in Fig. 6. The selection determines which two parents (task configuration sets) in the last generation will have offsprings in the next generation. The crossover operator determines how genes are exchanged between the parents to create those offsprings. The mutation allows for random alteration of genes. While the selection and crossover operators tend to increase the quality of the task execution in the new generation and force convergence, mutation tends to bring in divergence.

Parents Selection. A popular selection approach is the Roulette Wheel (RW) mechanism. In this method, if fitness $f(C_i)$ is the fitness of completed task performance in the candidate population, its probability of being selected is $P_i = \frac{f(C_i)}{\sum_{i=1}^{M} f(C_i)}$, where M is the number of tasks completed in the previous interval. This allows candidates with good fitness values to have a higher probability of being selected as parents. The selection module ensures reproduction of more highly fit candidates compared to the number of less fit candidates.

Crossover. A crossover function is used to cut the sequence of elements from two chosen candidates (parents) and swap them to produce two new candidates (children). As crossover operation is crucial to the success of Ant and it is also problem dependent, an exclusive crossover operation is employed for each individual. We implement relative fitness crossover [16] instead of absolute fitness crossover operation, because it moderates the selection pressure and controls the rate of convergence. Crossover operation is exercised on configuration candidates with a probability, known as crossover probability (P_c).

Mutation. The mutation function aims to avoid trapping in the local optimum by randomly mutating an element with a given probability. Instead of performing gene-by-gene mutation at each generation, a random number r is generated for each individual. If r is larger than the mutation probability (P_m), the particular individual undergoes the mutation process. Otherwise, the mutation operation involves replacing a

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

randomly chosen parameter with a new value generated randomly in its search space. This process prevents premature convergence of the population and helps Ant sample the entire solution space.

4.5 Aggressive Selection

The popular Roulette Wheel selection mechanism has a higher probability of selecting good candidates to be parents than bad ones. However, this approach still results in too many task evaluations, which in turn reduces the speed of convergence. Therefore, our selection procedure is more aggressive and deterministically selects good candidates to be parents. We use the following two strategies to accelerate the task-level parameter tuning.

Elitist Strategy. We found that good candidates are more likely to produce good offsprings. In this work, an elitist strategy is developed similar to the proposed GAs in study [16]. Elitism provides a means for reducing genetic drift by ensuring that the best candidate is allowed to copy their attributes to the next generation. Since elitism can increase the selection pressure by preventing the loss of low salience genes of candidates due to deficient selection pressure, it improves the performance with regard to optimality and convergence. However, the elitism rate should be adjusted suitably and accurately because high selection pressure may lead to premature convergence. The best candidate with highest fitness value in the previous generation will be preserved as one of the parents in the next generation.

Algorithm 2. Aggressive Selection Algorithm

1: /*Select the best configuration candidate C_{best}*/
2: $best = \arg\max_i [f(C_i)], i \in \{1, \ldots, M\}$
3: /*Select another configuration set from the candidates with fitness scores that exceed the mean by WR*/
4: $Avg_f = \frac{1}{M}\sum_{i=1}^{M} f(C_i)$
5: **if** $f(C_i) > Avg_f$ **then**
6: Select $C_{WR} = f(C_i)$ with possibility P_i;
7: **end if**
8: Use C_{best} and C_{WR} as parents.

Inferior Strategy. We also found that it is unlikely for two low fitness candidates to produce an offspring with high fitness. This is due to the fact that bad performance is often caused by a few key parameters and these bad settings continue to be inherited in real clusters. For instance, for an application that is both CPU and shuffle-intensive in a cluster with excessive I/O bandwidth and limited CPU resources, enabling compression of map outputs would stress the CPU and degrade application performance, regardless of others. The selection method should eliminate this configuration quickly. In order to quickly eliminate poor candidates, we calculate the mean fitness of the completed tasks for each generation and only select parents with fitness scores that exceed the mean.

Aggressive Selection Algorithm. Based on the above two aggressive selection strategies, the parents selection of the self-tuning reproduction is operated by an integrated selection algorithm. As shown in Algorithm 2, Ant firstly selects the best configuration candidate with the highest fitness in the last interval as one of the reproduction parents. Then it selects another configuration set from the candidates with fitness scores that exceed the mean by applying the Roulette Wheel approach. Finally, Ant generates the new generation configuration candidates by using the two selected candidates (i.e., C_{best} and C_{WR}) as the reproduction parents.

Furthermore, the aggressive selection strategies also reduce the impact of task skews during the tuning process. Long tasks due to data skews may add noises in the proposed GA-based task tuning. Taking the advantages of the aggressive selection, only the best configurations are possibly used to generate new configurations. It is unlikely that the tasks with skews would be selected as reproduction candidates. Thus, Ant would find the best configurations for the majority of tasks.

5 Moving Ant into the Cloud

Cloud computing offers users the ability to access large pools of computational and storage resources on demand. Developers by using the cloud computing paradigm are enabled to perform parallel data processing in a distributed and scalable environment with affordable cost and reasonable time. In particular, MapReduce deployment in the cloud allows enterprises to cost-effectively analyze large amounts of data without creating large infrastructures of their own [17]. Using virtual machines (VMs) and storage hosted by the cloud, enterprises can simply create virtual MapReduce clusters to analyze their data. However, tuning the jobs hosted in virtual MapReduce clusters is significant challenge to users. This is due to the fact that even the virtual node configured with the same virtual hardware could have varying capacity due to interferences from co-located users [18], [19]. When running Hadoop in a virtual cluster, finding nodes with the similar hardware capability is more challenging and complex than that in a physical cluster.

For moving Ant into virtual MapReduce clusters, it is apparently inaccurate to divide nodes into a number of homogeneous subclusters based on their static hardware configurations. Cloud offers elasticity to slice large, underutilized physical servers into smaller, parallel virtual machines, enabling diverse applications to run in isolated environments on a shared hardware platform. As a result, such a cloud environment leads to various interferences among the different applications. In this work, we focus on two representative interference scenarios in the cloud: stable interference and dynamic interference. We extend Ant to virtual MapReduce clusters in the cloud environments.

- In the first scenario of static interference, Ant only classifies VMs at the beginning of the adaptation process when interferences are relatively stable during the job execution process.
- In the second scenario of dynamic interference, the background interference could change and the performance of virtual nodes change over time. Thus, a re-clustering of the virtual nodes is needed to form new tuning groups. Ant is adapt to the variations of VM performance by periodically performing regrouping if significant performance changes in VMs are detected.

In the following, we describe how to classify a virtual MapReduce cluster into a number of homogeneous subclusters in the above two scenarios.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

5.1 Stable Interference Scenario

Ant estimates the actual capabilities of virtual nodes based on low-level resource utilizations of virtual resources. Previous study found that MapReduce jobs are mostly bottlenecked by the slow processing of a large amount of data [20]. Excessive I/O rate and a lack of CPU allocation are signs of slow processing. Ant characterizes a virtual node based on two measured performance statistics: I/O rate and CPU steal time. Both statistics can be measured at the `TaskTracker` of individual nodes.

Ant monitors the number of data bytes written to disk during the execution of a task. Since there is little data reuse in MapReduce jobs, the volume of writes is a good indicator of I/O access rate and memory demand. When the in-memory buffer, controlled by parameter `mapred.job.shuffle.merge.percent`, runs out or reaches the threshold number of map outputs `mapred.inmem.merge.threshold`, it is merged and spilled to disks. The spilled records are written to disks, which include both map and reduce spills. `TaskTracker` automatically records the data writing size and spilling duration of each task. We calculate the I/O access rate of individual nodes based on the data spilling operations.

The CPU steal time is the amount of time that the virtual node is ready to run but failed to obtain CPU cycles because the hypervisor is serving other users. It reflects the actual CPU time allocated to the virtual node and can effectively calibrate the configuration of virtual hardware according to the experienced interferences. We record the real-time CPU steal time by running the Linux `top` command on each virtual machine. `TaskTracker` reports the information to the master node in the cluster by the heartbeat connection.

Ant uses the k-means clustering algorithm to classify virtual nodes into configuration groups. Formally, given a set of VMs ($M_1, M_2, \ldots, M_m$), where each observation is a two-dimensional real vector (i.e., I/O rate and CPU steal time), k-means clustering aims to partition the m observations into $k(\leq n)$ sets $S = S_1, S_2, \ldots, S_k$ so as to minimize the within-group sum of squares:

$$arg \quad min \sum_{i=1}^{k} \sum_{M \in S_k} \|M - \mu_i\|^2. \tag{1}$$

Here μ_i is the mean of points in S$_i$. $\|M - \mu_i\|^2$ is a chosen distance(intra) measure between a data point M_m and the group centre μ_i. It is is an indicator of the distance of the mth VM from its respective group centers.

5.2 Dynamic Interference Scenario

The determination of k value in the proposed k-means classification approach is more difficult and complex when Ant is applied in a highly dynamic cloud environment. The real capacity of virtual nodes in the cluster will change over time as the background interference changes. Furthermore, virtual nodes could be migrated among different physical host machines in the cloud, which also has significant impact on the performance of virtual nodes. These observations motivate us to periodically perform re-grouping of the virtual nodes to form new tuning groups. Accordingly, we have to decide the number of homogeneous subclusters and the frequency of re-grouping the subclusters in a dynamic interference environment.

5.2.1 K Value Selection

As designed, the algorithm is not capable of determining the appropriate number of subclusters and it depends upon the user to identify this in advance. However, it is very difficult for Ant to set the number of groups in the dynamic interference scenario. In order to achieve the desired performance of Ant, it is necessary to find the number of groups for dynamic interferences among VMs on the runtime. Fixing the number of groups in a virtual cloud cluster may lead to poor quality of grouping and performance degradation. Thus, we propose a modified k-means method to find the number of groups based on the quality of the grouping output on the fly. It relies on the following two performance metrics to qualify the grouping of similar VMs in a virtual MapReduce cluster.

- *inter* is used to measure the separation of homogeneous VM groups. The term is the sum of the distances between the group centroids, which is defined as: $inter = \sum \|\mu_i - \mu_x\|^2, i, x \in k$.
- *intra* is used to measure the compactness of individual homogeneous VM groups. Here, we use the standard deviation as intra to evaluate the compactness of the data points in each group of VMs. It is defined as: $intra = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(M_i - \mu_i)^2}$.

As shown in Algorithm 3, users have the flexibility either to fix the number of groups or enter the minimum number of groups. In the former case, it works in the same way as the traditional k-means algorithm does. The value of k is empirically determined based on the historical workload records and MapReduce multi-wave behavior analysis. In the latter case, it lets the algorithm to compute the new number of groups by incrementing the group counter by one in each iteration until it satisfies the validity of grouping quality threshold, referring to line 14 in Algorithm 3.

Algorithm 3. Dynamic Grouping of VMs

1: **Inputs:** k: number of groups (initialize k = 2); M: number of VMs in a virtual cluster.
2: Randomly choose k VMs in the cluster, as k groups;
3: **repeat**
4:   **if** any other VM is not in a group **then**;
5:     assign the VM to the group according to Eq. (1);
6:     update the mean of the group;
7:   **end if**
8: **until** the minimum mean of the group is achieved;
9: **if** the number of groups is fixed **then**
10:   goto Output
11: **end if**
12: compute: $inter = \sum \|\mu_i - \mu_x\|^2, i, x \in k$
13: compute: $intra = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(M_i - \mu_i)^2}$
14: **if** $intra_k < intra_{k-1}$ and $inter_k > inter_{k-1}$ **then**
15:   $k \leftarrow k + 1$ and goto line 3 repeat;
16: **end if**
17: **Output:** a set of k groups.

5.2.2 Dynamic Re-Grouping

Since multi-tenancy interferences in the cloud are usually time-varying, grouping of homogeneous virtual nodes should be executed by k-means approach periodically.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

Intuitively, the frequency of the re-grouping is determined by two factors: the dynamics of multi-tenancy interference in the cloud and the size of the virtual MapReduce cluster.

- If the multi-tenancy interferences change frequently, re-grouping should be more frequent correspondingly so as to capture dynamic capacity changes of the virtual nodes.
- If the size of the virtual cluster is small, re-grouping should be less frequent. This is due to there is certain amount of time to find favorable task configurations after each re-grouping adjustment is done, which may weight out the benefit of re-grouping. On the contrary, re-grouping needs to be more frequent when the size of the virtual cluster is large.

Based on the above analysis, we formally formulate the dynamic capacity change of the virtual nodes based on the two performance metrics (i.e., $inter$ and $intra$) as

$$Dynamic(t) = \frac{intra(t)}{intra(t-1)} \times \frac{inter(t-1)}{inter(t)}. \quad (2)$$

Either the increase of $intra$ or the decrease of $inter$ means the dynamic capacity changes of VMs become significant. The growth of $intra$ means the compactness of individual homogeneous VM groups becomes relaxed. $inter$ reduction means the separation of homogeneous VM groups becomes unstable and regrouping becomes necessary. We periodically monitor the two performance metrics to reflect the interference in the cloud. Here, t is the interval to evaluate the dynamic capacity changes. We empirically set the value of t to one minute in the experiment. It is a tradeoff between the average task completion time and the fluctuation of interference in the cloud.

We formally design the dynamic re-grouping interval selection process as shown in Algorithm 4. The re-grouping interval (T_n) is initialized to 10 minutes at the beginning. It is then dynamically adjusted at runtime based on the dynamic capacity changes of the virtual nodes. As the dynamic increases, the re-grouping interval decreases, referring to line 4 in Algorithm 4. The item $Avg_{t \in T_n} Dynamic(t)$ represents the average dynamics during the previous re-grouping interval T_n. It aims to mitigate the impact of time-varying dynamic interference in the cloud.

Algorithm 4. Dynamic Re-Grouping Interval

1: **Inputs:** T_n: the interval of nth re-grouping (initialize T_1 = 10 minutes);
2: **repeat**
3: Collect statistics of $inter$ and $intra$ periodically;
4: Compute at the end of T_n: $T_{n+1} = \frac{T_n}{Avg_{t \in T_n} Dynamic(t)}$ according to Eq. (2);
5: **until** the job is completed;
6: **Output:** New re-grouping interval T_{n+1}.

6 IMPLEMENTATION

Hadoop Modification. We implemented Ant by modifying classes `JobTracker`, `TaskTracker` and `LaunchTaskAction` based on Hadoop version 1.0.3. We added a new interface `taskConf`, which is used to specify the configuration file of individual tasks while assigning them to slave nodes. Each set of task-level parameter set is tagged with its corresponding `AttemptTaskID`. Additionally, we added another new interface `Optimizer` to implement the GA optimization. During job execution, we created a method `taskAnalyzer` to collect the status of each completed task by using `TaskCounter` and `TaskReport`.

Ant Execution Process. At slave nodes, once a `TaskTracker` gets task execution commands from the `TaskScheduler` by calling `LaunchTaskAction`, it requires task executors to accept a `launchTask()` action from a local TaskTracker. Ant uses the `launchTask()` RPC connection to pass on the task-level configuration file description (i.e., `taskConf`), which is originally supported by the Hadoop. Ant creates a directory in the local file system to store the per-task configuration data for map/reduce tasks at `TaskTracker`. The directory is under the task working directory, and is tagged with `AttemptTaskID` which is obtained from `JobTracker`. Therefore, tasks can load their specified configuration items by accessing their task local file systems while initializing individual tasks by `Localizetask()`. Then after task localization, it kicks off a process to start a `MapTask` or `ReduceTask` thread to execute user-defined map and reduce functions.

Algorithm Implementation. We implemented the self-tuning algorithm to generate the configuration sets for the new generation tasks in each control interval (i.e., 5 minutes). The selection of the control interval is a trade-off between the parameter searching speed and average task execution time. If the interval is too long, it will take more time to find good configurations. If the interval is too short, the task with new configurations may not complete and no performance feedback can be collected. Thus, we choose a control interval of 5 minutes which is approximately two times of the average task execution time. The mutated value of a parameter is randomly chosen from its search space. Since our aggressive selection algorithm prunes poor regions, we can use an atypically high mutation rate (e.g., $p_m = 0.2$) without impacting convergence. The value of p_m is empirically determined. A cut point is randomly chosen in each parent candidate configuration and all parameters beyond that point are swapped between two parents to produce two children. We empirically set the crossover probability p_c to be 0.7.

7 EVALUATION ON A PHYSICAL CLUSTER

7.1 Experiment Setup

We evaluate Ant on a physical cluster [11] using three MapReduce applications from the PUMA benchmark [14] with different input sizes as shown in Table 3, which are widely used in evaluation of MapReduce performance by previous works [21]. We compare the performance of Ant with two other main competitors in practical use: Starfish [6], a job profiling based configuration approach from Duke university, and Rules-of-Thumb[1] (Heuristic), another representative heuristic configuration approach from industry leader Cloudera [12]. For reference, we normalize the Job Completion Time achieved by various approaches to the JCT achieved by

1. Cloudera recommends a set of configuration items based on its industry experience, e.g., `io.sort.record.percent` is recommended to set as $\frac{16}{16+avg-record-size}$, which is based on the average size of map output records. More rules are available in [12].

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
The Characteristics of Benchmark Applications Used in Our Experiments

Category	Type	Label	Input size (GB)	Input data	# Maps	# Reduces
Wordcount	CPU intensive	J1/J2/J3	100/300/900	Wikipedia	400/1,200/3,600	14/ 14/ 14
Grep	I/O intensive	J4/J5/J6	100/300/900	Wikipedia	400/1,200/3,600	14/ 14/ 14
Terasort	I/O intensive	J7/J8/J9	100/300/900	TeraGen	400/1,200/3,600	14/ 14/ 14

the Hadoop stock parameter settings. Unless otherwise specified, we use the stock configuration setting of Hadoop implementation for the other items that are not listed in the preliminary study [11]. Note that both Heuristic and Starfish always maintain identical configuration files for job executions as described in Section 2. For fairness, the cluster-level and job-level parameters for all approaches (including the baseline stock configuration) in the experiments are set to suggested values by the rules of thumb from Cloudera [12]. For example, we roughly set the value of `mapred.reduce.tasks` (the number of reduce tasks of the job) to 0.9 times the total number of reduce slots in the cluster.

7.2 Effectiveness of Ant

Reducing Job Completion Time. Fig. 7 compares various job completion times achieved by Heuristic, Starfish and Ant, respectively. The results demonstrate that all of these configuration approaches improve the job completion times more or less when compared with the performance achieved by Hadoop stock parameter setting (Stock). Fig. 7 shows that Ant improves the average job completion time by 31, 20 and 14 percent compared with Stock, Heuristic and Starfish on the physical cluster, respectively. This is due to the fact that Stock, Heuristic and Starfish all rely on a unified and static task-level parameter setting. Such unified configuration is apparently inefficient in the heterogeneous cluster. The results also reveal that Starfish is more effective than Heuristic on the physical cluster since Starfish benefits from its job profiling capability. The learning process of Starfish is more accurate than the experience based tuning of Heuristic to capture the characteristic of individual jobs.

Impact of Job Size. Fig. 7 shows that Ant slightly reduces the completion time of small jobs compared with stock Hadoop, e.g., J1, J4 and J7. In contrast, Ant is more effective for large jobs, e.g., J3, J6 and J9. This is due to the fact that small jobs usually have short execution times and the self-tuning process can not immediately find the optimal configuration solutions. Thus, such small jobs are not favored by Ant.

Impact of Workload Type. Fig. 7 reveals that Ant reduces the job completion time of I/O intensive workloads, i.e., *Grep* and *Terasort*, 10 percent more than that of CPU intensive workloads, i.e., *Wordcount*. This is due to the fact that Ant focuses on the task-level I/O operation parameter tuning and accordingly it affects more for I/O intensive workloads.

Overall, Ant achieves consistently better performance than Heuristic and Starfish do on the physical Hadoop cluster. This is due to its capability of adaptively tuning task-level parameters while considering various workload preferences and heterogeneous platforms.

7.3 Ant Searching Process

In order to take a close look of Ant searching process, Fig. 8 depicts the parameter `io.sort.record.percent` searching path from two representative machines (i.e., Atom and T110) in the physical cluster. Fig. 8a shows that Atom spends around 20 minutes to find a stable parameter range. In contrast, Fig. 8b shows that T110 takes only 7 minutes due to the fact that there are three T110 machines in the cluster which provide three times concurrent running tasks of Atom's. More parallel task executions mean there are more opportunities to learn the characteristics of the running job on the cluster in the same time period.

8 Evaluation on Virtual Clusters

8.1 Experiment Setup

We built a virtual cluster in our university cloud. VMware vSphere 5.1 was used for server virtualization. VMware vSphere module controls the CPU usage limits in MHz allocated to VMs. We created a pool of VMs with different hardware configurations from the virtualized blade server cluster and run them as Hadoop slave nodes. All VMs ran Ubuntu Server 12.04 with Linux kernel 3.2. The cluster-level configurations for Hadoop are the same as those in the physical cluster (Section 7.1). The number of reduce tasks is set to 42, which is 0.9 times the number of the available reduce slots in the cluster.

Stable Interference Scenario. Please refer to the preliminary study [11] for the effectiveness of Ant with stable interference.

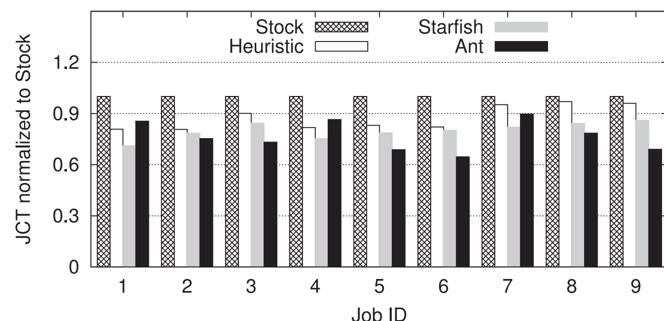


Fig. 7. Job completion time on the physical cluster.

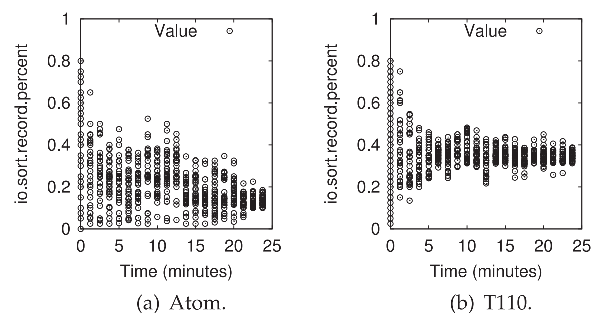


Fig. 8. Task-level parameter search process.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

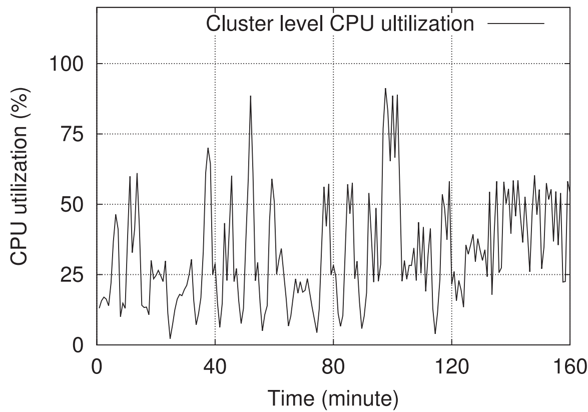


Fig. 9. Dynamic interference in the cloud.

Dynamic Interference Scenario. In the virtual cluster there are 24 VMs, each with 2 vCPU, 4 GB RAM and 80 GB hard disk space, which are hosted on blade servers in our cloud. Unlike the stable interference scenario, we do not run any specific applications in the cloud as the dynamic interference producer. Instead, there are various applications (e.g., scientific computations and multi-tier Web applications) hosted in our university cloud over time. These different type and time-varying applications play the role of dynamic interference producer in the cloud environment. As shown in Fig. 9, we record the cluster level CPU utilization to demonstrate the interference in the cloud. We set the initialized value of k to 2 in Algorithm 3 and then dynamically divide the virtual cluster into a few subclusters by using the modified k-means clustering approach. The re-grouping interval is dynamically adjusted based on the Algorithm 4 so that the grouping of virtual nodes is updated dynamically to capture the interference changes in the cloud.

We firstly demonstrate the performance improvement achieved by Ant on the virtual cluster with dynamic interferences. We then evaluate the effectiveness of the proposed re-grouping approach in the dynamic interference environment. Finally, we analyze the effectiveness of the dynamic re-grouping interval selection.

8.2 Effectiveness of Ant under Dynamic Interference

Fig. 10 compares various job completion times achieved by Heuristic, Starfish and Ant in the cloud with dynamic interference, respectively. Fig. 10 demonstrates that all of these configuration approaches improve the job completion time compared with that achieved by stock Hadoop parameter setting approach (Stock). The result shows that Ant improves the average job completion time by 20, 15 and 11 percent compared with Stock, Heuristic and Starfish on the physical cluster, respectively. As described in the physical cluster scenario, Stock, Heuristic and Starfish all rely on a unified and static task-level parameter setting. Such unified configurations are apparently inefficient in a virtual cluster with various interferences. In the cloud environment, task configurations should be changed in response to the changes of interferences. The results also reveal that Heuristic is more effective than Starfish on the virtual cluster. Although the learning based Starfish is more accurate than the experience based Heuristic in the physical cluster, it fails to capture the characteristic of various dynamic interferences in the cloud environment.

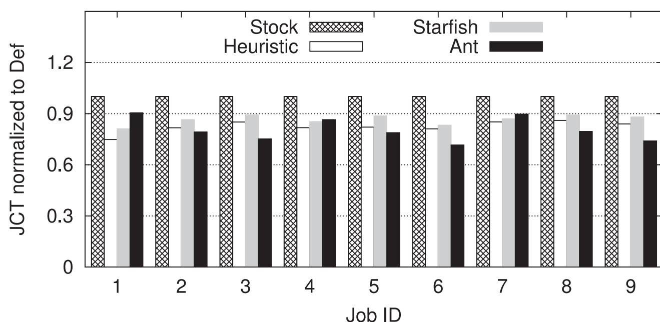
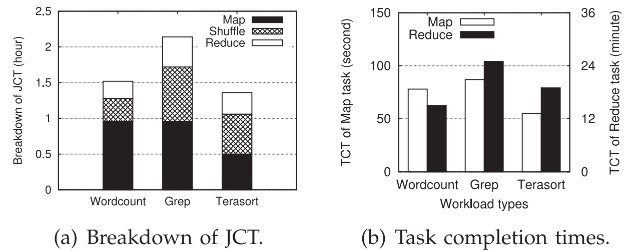


Fig. 10. Job completion time on the virtual cluster with dynamic performance interference.

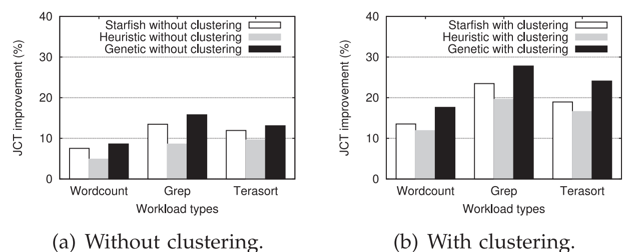


(a) Breakdown of JCT. (b) Task completion times.

Fig. 11. Impact on task-level performance.

Fig. 11 shows the detailed impact on task-level performance while applying Ant in the cloud environment. Fig. 11a depicts the breakdown of job completion time of the three jobs (i.e., J3, J6 and J9) used in the experiment. It reveals that Wordcount is map-intensive while Terasort and Grep are shuffle & reduce-intensive. Fig. 11b shows the average task completions time of map tasks and reduce tasks in the experiment. The result demonstrates that map tasks are relatively small compared to reduce tasks. This is due to the fact that the number of reduce tasks is configured to 0.9 times of the total number of reduce slots in the cluster. It aims to complete the execution of reduce tasks in one waive to avoid the overhead caused by data shuffle.

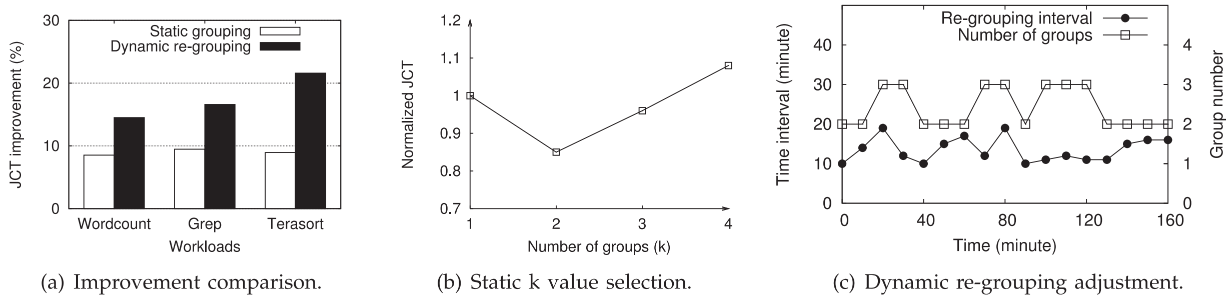
Fig. 12 shows the job completion time improvement achieved by Genetic with/without clustering, Starfish with/without clustering, and Heuristic with/without clustering approaches in a single heterogeneous cluster. The results demonstrate approaches with clustering achieve better performance improvement than approaches without clustering in heterogeneous environments. We find that Genetic with clustering achieves slight performance improvement compared to the other two approaches. This is due to the fact that



(a) Without clustering. (b) With clustering.

Fig. 12. Comparsion by applying different tuning approaches in a heterogeneous cluster.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.



(a) Improvement comparison. (b) Static k value selection. (c) Dynamic re-grouping adjustment.

Fig. 13. Performance improvement comparison between static grouping and dynamic re-grouping approach.

the main contribution of Ant relies on the dynamic clustering and task-level adaptive configuration in heterogeneous environments.

8.3 Effectiveness of Subcluster Re-Grouping

Fig. 13a compares the job completion time improvement achieved by Ant with the dynamic subcluster re-grouping and static grouping capability. It shows that Ant with the dynamic re-grouping capability can outperform Ant with static grouping capability by almost 100 percent in terms of the job completion time improvement. This is due to the benefit of re-grouping virtual nodes into a number of homogeneous subclusters so as to mitigate the impact of interference on task tuning. In the approach of Ant without re-grouping, it uses a static number of groups that is similar with the stable interference scenario. Ant only groups the virtual nodes at the beginning of the task tuning process in the dynamic interference environment. Thus, Ant without subcluster re-grouping capability cannot capture time-varying interference in the cloud. Fig. 13b shows the impact of the number of groups on the job completion time under the static grouping approach. We empirically select the static number of group (i.e., $k = 2$) for comparison with the dynamic subcluster re-grouping capability.

Fig. 13c shows that both the number of groups and the re-grouping interval are dynamically adjusted based on the time-varying interference in the cloud. Ant periodically collects the dynamic capacity changes of the virtual nodes and then re-groups virtual nodes according to Algorithms 3 and 4. Fig. 13c depicts the dynamic tuning process of Ant in a 160-minute time window, which is corresponding to the time-varying interference scenario as shown in Fig. 9. The number of groups is initialized as two at the beginning stage. It is then dynamically tuned based on the dynamic capacity changes. The result shows that the number of groups keeps stable at two when the interference is relatively stable, such as in the period between 120th and 160th minutes. However, the number of groups fluctuates when the interference changes frequently, such as in the period around 100th minute.

At the same time, the interval of re-grouping is adaptively changed with the interference changes at the runtime. When the cloud interference fluctuates significantly (e.g., 90th to 110th minutes), the re-grouping interval becomes small so that Ant can capture the capacity changes of the virtual nodes in the cluster. The re-grouping interval is larger when the interference changes more frequently in the could. It represents system performance sensitivity of the interference dynamics in our university cloud.

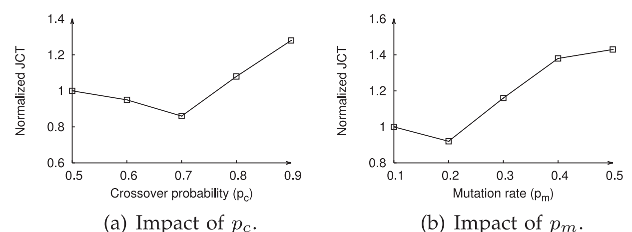
8.4 Sensitivity of GA Parameter Selection

We change the values of crossover probability and mutation rate to study their impact on the performance improvement in terms of job completion time. Fig. 14a shows that the job completion time initially decreases as the crossover probability increases. However, increasing the crossover probability further leads to performance degradation. This tells that a very large crossover probability may lead to job completion time deterioration. Thus, we empirically set the crossover probability to 0.7 in the experiment. It is a tradeoff between the search speed and the job completion time improvement. Fig. 14b shows tuning the mutation rate has the similar phenomenon with the crossover probability in the experiment. A large mutation rate (e.g., 0.5) leads significant performance deterioration due to the instability of task-level configuration. Thus, we empirically set the mutation rate to 0.2 without impacting convergence in the experiment.

9 Related Work

Heterogeneous Environment. As heterogeneous hardware is applied to Hadoop clusters, how to improve MapReduce performance in heterogeneous environments attracts much attention [1], [2], [22], [23]. Ahmad et al. [1] identified key reasons for MapReduce poor performance on heterogeneous clusters. Accordingly, they proposed an optimization based approach, Tarazu, to improve MapReuce performance by communication-aware load balancing. Zaharia et al. [2] designed a robust MapReduce scheduling algorithm, LATE, to improve the completion time of MapReduce jobs in a heterogeneous environment. They paid little attention to optimizing Hadoop configurations, which has a significant impact on the performance of MapReduce jobs, especially in a heterogeneous Hadoop cluster.

Parameter Configuration. Recently, a few studies start to explore how to optimize Hadoop configurations to improve job performance. Herodotou et al. [7] proposed several automatic optimization based approaches for MapReduce parameter configuration to improve job performance.



(a) Impact of p_c. (b) Impact of p_m.

Fig. 14. Sensitivity of the crossover probability and mutation rate tuning.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

Kambatla et al. [24] presented a Hadoop job provisioning approach by analyzing and comparing resource consumption of applications. It aimed to maximize job performance while minimizing the incurred cost. Lama and Zhou designed AROMA [8], an approach that automated resource allocation and configuration of Hadoop parameters for achieving the performance goals while minimizing the incurred cost. Herodotou et al. proposed Starfish [6], an optimization framework that hierarchically optimizes from jobs to workflows by searching for good parameter configurations. These approaches mostly rely on the default Hadoop framework and configure the parameters by static settings. They are often not effective when the cluster platform becomes heterogeneous.

MapReduce in the Cloud. Currently, there are several options for using MapReduce in the cloud environments, such as using MapReduce as a service, setting up one's own MapReduce cluster on cloud instances, or using specialized cloud MapReduce runtimes that take advantage of cloud infrastructure services. Guo et al. [25] designed FlexSlot, an effective yet simple extension to the slot-based Hadoop task scheduling framework. It adaptively changes the number of slots on each virtual node to promote efficient usage of resource pool in cloud environment. Chiang et al. [20] presented TRACON, a novel task and resource allocation control framework that mitigated the interference effects from concurrent data-intensive applications. Ant differentiates itself from those efforts through its capability of adaptive task-level tuning to achieve performance optimization.

10 Conclusion and Future Work

Although a unified design framework, such as MapReduce, is convenient and easy to use for large-scale parallel and distributed programming, it ignores the differentiated needs in the presence of various platforms and workloads. In this paper, we tackle a practical yet challenging problem of automatic configuration of large-scale MapReduce workloads in heterogeneous environments. We have proposed and developed a self-adaptive task-level tuning approach, Ant, that automatically finds the optimal settings for individual jobs running on heterogeneous nodes. In Ant, tasks are customized with different settings to match the capabilities of heterogeneous nodes. It works best for large jobs with multiple rounds of map task execution. Our experimental results demonstrate that Ant can improve the average job completion time on a physical cluster by 31, 20, and 14 percent compared to stock Hadoop, customized Hadoop with industry recommendations, and a profiling-based configuration approach, respectively. Experimental results on two virtual cloud clusters with varying multi-tenancy interferences show that Ant improves the average job completion time by 20, 15, and 11 percent compared to Stock, Heuristic and Starfish, respectively. Ant can be deployed to different types of clusters, and thus is flexible and adaptive.

Our method Ant can be extended to other frameworks such as Spark, though some additional effort is needed. Different from Hadoop, which executes individual tasks in separate JVMs, Spark uses executors to host multiple tasks on worker nodes. To extend Ant to Spark, we need to dynamically change executor sizes without restarting a launched job. Since running Spark on another generic cluster management middleware, such as YARN, becomes increasingly popular, it is possible to enable malleable executors using resource containers. As such, Ant can monitor the completion times of individual tasks and use such information as feedback to determine the optimal size of Spark executors. In the future, we also plan to extend Ant to multi-tenancy public cloud environments such as Azure and EC2.

Acknowledgments

This research was supported in part by U.S. National Science Foundation research grants CNS-1422119, CNS-1320122, CNS-1217979, and NSF of China research grant 61328203. A preliminary version of the paper appeared in [11]. The authors are grateful to the editor and anonymous reviewers for their valuable suggestions for revising the manuscript. Xiaobo Zhou is a corresponding author.

References

[1] F. Ahmad, S. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *Proc. Int. Conf. Architecture Support Program. Language Operating Syst.*, 2012, pp. 61–74.

[2] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2008, pp. 29–42.

[3] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and deadline-aware job scheduling in dynamic hadoop clusters," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2015, pp. 956–965.

[4] D. Cheng, P. Lama, C. Jiang, and X. Zhou, "Towards energy efficiency in heterogeneous hadoop clusters by adaptive task assignment," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 359–368.

[5] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proc. ACM Symp. Cloud Comput.*, 2011, pp. 18:1–18:14.

[6] H. Herodotou, et al., "Starfish: A self-tuning system for big data analytics," in *Proc. Conf. Innovative Data Syst. Res.*, 2011, pp. 261–272.

[7] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of MapReduce programs," in *Proc. Int. Conf. Very Large Data Bases*, 2011, pp. 1111–1122.

[8] P. Lama and X. Zhou, "AROMA: Automated resource allocation and configuration of MapReduce environment in the cloud," in *Proc. Int. Conf. Autonomic Comput.*, 2012, pp. 63–72.

[9] M. Li, et al., "MRONLINE: MapReduce online performance tuning," in *Proc. ACM Symp. High-Performance Parallel Distrib. Comput.*, 2014, pp. 165–176.

[10] T. White, *Hadoop: The Definitive Guide*, 3rd ed. *O'Reilly Media/Yahoo Press*, Sebastopol, California, 2012.

[11] D. Cheng, J. Rao, Y. Guo, and X. Zhou, "Improving mapreduce performance in heterogeneous environments with adaptive task tuning," in *Proc. ACM/IFIP/USENIX Int. Middleware Conf.*, 2014, pp. 97–108.

[12] Cloudera, "Configuration parameters," 2012. [Online]. Available: http://blog.cloudera.com/blog/author/aaron/

[13] MapR, "The executives guide to big data," 2013. [Online]. Available: http://www.mapr.com/resources/white-papers

[14] PUMA, "Purdue mapreduce benchmark suite," 2012. [Online]. Available: https://engineering.purdue.edu/ puma/datasets.htm

[15] Y. Guo, J. Rao, and X. Zhou, "iShuffle: Improving hadoop performance with shuffle-on-write," in *Proc. Int. Conf. Autonomic Comput.*, 2013, pp. 107–117.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi objective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[17] B. Igou, "User survey analysis: Cloud-computing budgets are growing and shifting; traditional it services providers must prepare or perish," *Gartner Report*, G00205813, 2010.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.

[18] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for MapReduce in a cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1265–1279, May 2015.

[19] B. Sharma, T. Wood, and C. R. Das, "Hybridmr: A hierarchical MapReduce scheduler for hybrid data centers," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 102–111.

[20] R. C. Chiang and H. H. Huang, "Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. Int. Conf. High Performance Comput., Netw., Storage Anal.*, 2011, pp. 1–12.

[21] B. Cho, et al., "Natjam: Eviction policies for supporting priorities and deadlines in MapReduce clusters," in *Proc. ACM Symp. Cloud Comput.*, 2013, pp. 6:1–6:17.

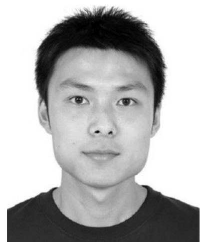
[22] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Elastic power-aware resource provisioning of heterogeneous workloads in self-sustainable datacenters," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 508–521, Feb. 2016.

[23] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-aware workload placement and migration in distributed sustainable datacenters," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2014, pp. 307–316.

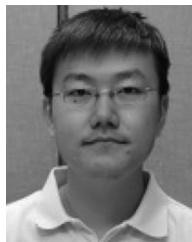
[24] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in *Proc. USENIX HotCloud Workshop*, 2009, Art. no. 22.

[25] Y. Guo, J. Rao, C. Jiang, and X. Zhou, "Moving MapReduce into the cloud with flexible slot management," in *Proc. Int. Conf. High Performance Comput., Netw., Storage Anal.*, 2014, pp. 959–969.

Dazhao Cheng received the BS and MS degrees in electronic engineering from Hefei University of Technology, in 2006 and the University of Science and Technology of China, in 2009, respectively, and the PhD degree from the University of Colorado, Colorado Springs, in 2016. He is currently an assistant professor in the Department of Computer Science, University of North Carolina, Charlotte. His research interests include cloud computing and big data prcoessing. He is a member of the IEEE.



Jia Rao received the BS and MS degrees in computer science from Wuhan University, in 2004 and 2006, respectively, and the PhD degree from Wayne State University, in 2011. He is currently an assistant professor in the Department of Computer Science, University of Colorado, Colorado Springs. His research interests include the areas of distributed systems, resource auto-configuration, machine learning, and CPU scheduling on emerging multi-core systems. He is a member of the IEEE.



Yanfei Guo received the BS degree in computer science and technology from Huazhong University of Science and Technology, China, in 2010, and the PhD degree in computer science from the University of Colorado, Colorado Springs, in 2015. He is currently a postdoc fellow in the Argonne National Lab. His research interests include cloud computing, big data processing, MapReduce, and HPC. He is a member of the IEEE.

Changjun Jiang received the PhD degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995. Currently he is a professor in the Department of Computer Science, Tongji University, Shanghai. He is also the director of the Professional Committee of Petri Net of China Computer Federation and the vice director of the Professional Committee of Management Systems of China Automation Federation. His research interests include concurrent theory, Petri net, and intelligent transportation systems. He is a member of the IEEE.

Xiaobo Zhou received the BS, MS, and PhD degrees in computer science from Nanjing University, in 1994, 1997, and 2000, respectively. Currently he is a professor and the chair in the Department of Computer Science, University of Colorado, Colorado Springs. His research lies broadly in computer network systems, specifically, cloud computing and datacenters, bigdata parallel and distributed processing, autonomic and sustainable computing, scalable Internet services, and architectures. He received the NSF CAREER Award in 2009. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: Wuhan University. Downloaded on September 16,2022 at 06:44:18 UTC from IEEE Xplore. Restrictions apply.