Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Multi-Tier Resource Allocation for Data-Intensive Computing ☆

Thomas Ryan [a], Young Choon Lee [b,*]

[a] School of Information Technologies, The University of Sydney, NSW, 2006, Australia
[b] Department of Computing, Macquarie University, NSW, 2109, Australia

ARTICLE INFO

Article history:
Received 22 December 2014
Received in revised form 9 February 2015
Accepted 2 March 2015
Available online xxxx

Keywords:
Resource allocation
Data-intensive computing
Cloud computing
Big data
Application heterogeneity
MapReduce

ABSTRACT

As distributed computing systems are used more widely, driven by trends such as 'big data' and cloud computing, they are being used for an increasingly wide range of applications. With this massive increase in application heterogeneity, the ability to have a general purpose resource management technique that performs well in heterogeneous environments is becoming increasingly important.
In this paper, we present Multi-Tier Resource Allocation (MTRA) as a novel fine-grained resource management technique for distributed systems. The core idea is based on allocating resources to individual tasks in a tiered or layered approach. To account for heterogeneity, we propose a dynamic resource allocation method that adjusts resource allocations to individual tasks on a cluster node based on resource utilisation levels. We demonstrate the efficacy of this technique in a data-intensive computing environment, MapReduce data processing framework in Hadoop YARN. Our results demonstrate that MTRA is an effective general purpose resource management technique particularly for data-intensive computing environments. On a range of MapReduce benchmarks in a Hadoop YARN environment, our MTRA technique improves performance by up to 18%. In a Facebook workload model it improves job execution times by 10% on average, and up to 56% for individual jobs.

1. Introduction

As the scale and size of applications continues to increase with the explosive growth in data volume (dubbed 'Big Data'), distributed processing/computing has become rather essential. The availability of virtually unlimited resource capacity with cloud computing has greatly enabled such distributed computing. As a result, a wide range of distributed systems has been developed. Some popular examples are MapReduce data processing framework [1], Pregel graph processing system [2] and Montage astronomical image mosaic engine [3,4]. Applications in these distributed systems exhibit much heterogeneity in terms particularly of resource usage characteristics, e.g., CPU-intensive applications and data-intensive applications.

Within distributed systems, resource allocation and management is an ongoing research concern; it is well known that improving resource allocation mechanisms can result in considerable real-world improvements in performance and efficiency. However, different applications have different resource requirements, meaning that effective resource management in heterogeneous environments is much more difficult. While there has been a large amount of research into resource allocation, most resource allocation strategies do not account for application heterogeneity. In particular, the majority of classic optimisation techniques for resource management assume homogeneity in either application or resource, or both. And thus, they often do not perform optimally in heterogeneous environments.

In this paper, we address the problem of fine-grained resource allocation to distributed systems where jobs are composed of many small tasks taking into account application heterogeneity. The traditional approach to resource management in these systems has been to divide resources into logical partitions (called 'slots' or 'containers'), and allocate tasks to partitions [1,5]. However, since different jobs have different resource usage characteristics, this approach can lead to both resource under-utilisation and resource contention, resulting in decreased performance. A general purpose, scheduler-independent solution to this problem is highly desirable, especially as we see increasing heterogeneity of both applications and execution environments.

To this end, we develop Multi-Tier Resource Allocation (MTRA) as a novel resource management technique that dynamically adjusts resource allocations to heterogeneous, individual tasks in distributed systems. The dynamic adjustments are based on the

☆ This article belongs to BDA-HPC.

* Corresponding author.
E-mail addresses: trya3473@uni.sydney.edu.au (T. Ryan), young.lee@mq.edu.au (Y.C. Lee).

http://dx.doi.org/10.1016/j.bdr.2015.03.001
2214-5796/

resource requirements of each task as well as current levels of resource utilisation and resource contention on each node in a given distributed computing system, or simply cluster. The idea is that a common set of resources can be dynamically multiplexed in several fine-grained resource allocation tiers to enable multiple tasks with different resource usage characteristics to "harmoniously" share the resource set. Rather than relying on a centralised scheduler, resource allocations are adjusted locally on each node, allowing for very fine-grain control. This distributed resource management approach can be combined with any existing scheduler. It also decreases scheduler complexity, allowing for greater scalability. Note that our MTRA is a resource management technique underneath any schedulers that deal with the globalisation of the resource allocation. The current implementation of MTRA primarily deals with CPU and IO resources although it potentially capable of dealing with other resources, such as network resources.

Our main focus is on MapReduce applications within a Hadoop YARN environment. Resource management strategies in Hadoop are based on the idea of logical partitions of resources. Traditionally, these have been statically configured, a process which is itself an ongoing research issue. Since different tasks require different amounts and kinds of resources, configuration in heterogeneous environments is even more problematic. Our preliminary solution to this problem is Local Resource Shaper (LRS) [6] that enables local resources (CPU core and disks) to be shared primarily by two tiers allowing dynamic movement of tasks between resource allocation tiers. While this model is shown to improve resource utilisation while minimising resource contention in Classic Hadoop, we show that it does not account well for task heterogeneity. With regard to both application and environment heterogeneity, MTRA enables optimal performance by adjusting allocations to maximise resource utilisation and minimise resource contention for the resource requirements of each task.

Our evaluation shows that by accounting for heterogeneity, our MTRA technique improves system performance. MTRA is comparable or better than the previous best resource management alternative for a range of MapReduce benchmarks with different resource usage characteristics, including outperforming the two-tier allocation model [6]. For individual MapReduce benchmarks, we observe performance improvements of up to 18% compared to Hadoop YARN. We show that for a workload model based on a Facebook production workload, MTRA reduces individual job execution times by 10% on average and up to 56% for individual jobs in the workload. These results prove that MTRA improves performance by a significant margin in real-world, heterogeneous environments.

The rest of this paper is organised as follows. Section 2 gives background. Section 3 presents our multi-tier resource allocation technique and describes the application of MTRA to MapReduce in Hadoop YARN. Section 4 presents our evaluation on the efficacy of MTRA. Section 5 discusses the related work. Finally, Section 6 concludes the paper.

2. Background

In this section, we begin by some background on big data and cloud computing, and then provide essential works and application used in this paper.

2.1. Big data and cloud computing

The recent trend of so-called 'Big Data' is expected to be rather norm as the volume of data exponentially increases literally in every area—business, science, and daily life to name a few. Today, some claim that data (more specifically, data-intensive science/computing) are the fourth paradigm in scientific research after experimentation/observation, theory, and computational simulation [7]. Efficient data storage and timely data processing is of great practical importance. Clearly, this requires both massive storage and processing capacity. While some data processing deals with simple retrieval or search (IO-intensive), other case involves much computation on data (mixture usage of IO and CPU resources); hence, application heterogeneity.

Cloud computing has emerged as a new computing paradigm with its strengths in elasticity and pay-as-you-go pricing, and it is a viable solution to many ICT services including big data processing. Cloud computing in this context refers to the Infrastructure-as-a-Service model, such as that provided by Amazon Elastic Compute Cloud (EC2) [8] or Google Compute Engine [9]. This model has many advantages, including elasticity, scalability, and using a pay-as-you-go pricing system [10]. All of these properties have implications for cluster environments, in terms of economics (only paying for what you need) and performance (such as the shared-tenant hardware and unpredictable datacentre loads). The use of public cloud providers is associated with performance overheads due to a range of factors. Individual nodes, or 'instances', are based in virtual machines (VMs) which do not necessarily correspond to physical machines. This representation of instances provides a lot of control, however information such as the network topology within the datacentre is unavailable; this has means we cannot guarantee our nodes will be provisioned on the same server or rack, leading to inconsistent communication times between nodes. Additionally, it is likely that instances will "statically" share a physical machine with other cloud users, with potential performance impacts if an instance is co-located with a resource-greedy neighbour.

2.2. MapReduce and Hadoop

The MapReduce framework is a widely used programming paradigm for distributed environments [1]. MapReduce provides an abstraction away from the details of parallelising computation; the framework automatically divides a job into individual tasks, handles scheduling of individual tasks, distributes data and deals with machine failures. The basic MapReduce model expresses computations as a 'Map' and a 'Reduce' function. Hadoop is a framework for the execution of MapReduce jobs. Classic Hadoop has been widely used and studied since its release, but we focus on the more recently developed Hadoop YARN [5]. Both Classic Hadoop and Hadoop YARN use the idea of dividing resources into logical partitions (called 'slots' and 'containers' respectively) which are assigned to executing tasks.

2.3. Local resource shaper for MapReduce

The Local Resource Shaper (LRS) for MapReduce [6] modifies the slot-based resource allocation approach used in Classic Hadoop. Rather than statically configured Map and Reduce slots, LRS introduces the idea of a dual purpose task slot. LRS 'shapes' task resource usage by allocating resources to tasks in a tier-based model. Tasks are split into Active and Passive tiers, with resources allocated such that the Active task uses as much resources as possible to maintain its original usage, while the Passive task uses resources unused by the Active task. In Classic Hadoop, resources are allocated based on a slot model, with each slot representing a partition of resources. LRS pairs slots such that for each Active task (i.e., slot), there is an associated Passive task. This means that within each pair, the two tasks have complementary resource usage; for example, while the Active task waits on I/O operations to complete, the Passive task is able to use the otherwise wasted CPU resources. This approach is shown to significantly increase resource utilisation while minimising resource contention, resulting

Please cite this article in press as: T. Ryan, Y.C. Lee, Multi-Tier Resource Allocation for Data-Intensive Computing, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.03.001

in improved performance. We will demonstrate in Section 3 that this technique improves performance less in Hadoop YARN than in Classic Hadoop since Hadoop YARN already uses multi-purpose resource containers, allowing for significantly higher baseline resource utilisation [5].

3. Multi-tier resource allocation

In this section, we present our Multi-Tier Resource Allocation (MTRA) technique and detail its implementation in Hadoop YARN. Unlike previous concrete approaches that focus on the exclusiveness and isolation of resource use between co-located applications by explicitly controlling resource usage [11–13], MTRA strives for the "organification" of resource sharing by interlacing resource usage providing non-intrusive resource sharing. Unless the resource usage of co-located applications perfectly complement each other, when using previous solutions, resource contention and performance variability is inevitable. An overview of the architecture is given in Fig. 1.

3.1. Design of MTRA

While the capacity of computing resources becomes increasingly large, many applications are unable to fully/effectively use these powerful resources. Specifically, a particular application running on a set of dedicated resources mostly not use these resources all the time; and, the co-location of this application with one or more applications may result in performance interference due to their heterogeneity in resource usage patterns. Our multi-tier resource allocation (MTRA) technique essentially exploits this application heterogeneity, more precisely heterogeneity in their resource usage characteristics. MTRA extends the preliminary two-tier design of LRS with an additional 'Frozen' tier and a more streamlined dynamic resource allocation mechanism. The design of three MTRA tiers is shown in the Resource Containers part of Fig. 1.

In LRS's two-tier system, resource allocation groups consisted of one Active and one Passive task; we add one Frozen task to each group, although other configurations are possible. The number of tiers can largely be determined based on types of application (more precisely, resource usage patterns) of interest. As MapReduce tasks typically consume more than 50% of a CPU resource [14], our original design of LRS adopted two tiers. However, we have observed that there are still some resource capacity frequently idle; hence, we introduce the addition of Frozen tier to MTRA. As in LRS, we use Linux `cgroups` for fine-grained resource allocation and performance isolation. MTRA allocates CPU and I/O resources to Active and Passive tasks in the ratio 100:1 for the maximum usage (100% usage) of 101 (the summation of ratio values). This ratio dictates the relative resource usage between Active/Passive slots and it can be interpreted as 99% and 1% of resource usage for Active and Passive, respectively; and, Frozen tasks receive no resources by default using the `cgroups` Frozen group. This means execution of tasks on the Frozen tier is effectively paused.

When the monitoring process detects resource under-utilisation, a Frozen task is dynamically allocated resources to allow it to execute on the Passive tier. Conversely, if resource contention passes a certain threshold, resource allocations are again dynamically adjusted to pause the execution of a Passive task, or to 'freeze' it. This technique allows us to respond to changes in resource utilisation in a much finer-grained manner than a centralised scheduler-based system. Its most significant advantages are in accounting for heterogeneity. Dynamic resource allocation allows for allocations to change based on the resource requirements of the individual tasks on individual nodes, resulting in maximising resource utilisation and minimising resource contention in heterogeneous environments.

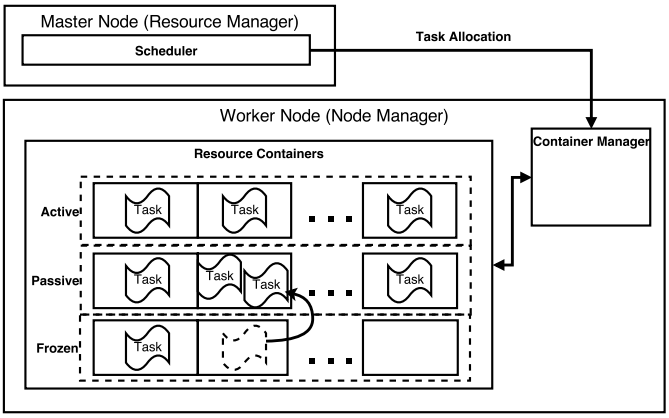

Fig. 1. Architecture of the MTRA implementation in Hadoop YARN. The arrow shows task movement between Frozen and Passive tiers.

3.2. MTRA implementation for MapReduce

It is a known problem in MapReduce and similar systems that the best slot or container configuration is dependent on job characteristics, with no single best static solution to this problem [6,1,5]. To solve this, MTRA dynamically adjusts container resource allocation to resource requirements of executing tasks by monitoring resource utilisation.

As MTRA is a general resource management technique, it can be scheduler obliviously incorporated. Our Hadoop integration mainly concerns with the Hadoop YARN Container Manager interface (Fig. 1). In particular, the Hadoop YARN Container Manager interface to Linux `cgroups` has been extended to handle MTRA with the additional Frozen tier. The Hadoop YARN Container Manager thread has also been extended to monitor system resource usage and dynamically adjust the number of Passive tasks in response to resource under-utilisation or resource contention, i.e., most MapReduce tasks consume 50% or more of a CPU resource [14]. The formula for calculating the number of Passive tasks (Eq. (1)) is the same as that proposed by [6]. As the dynamic adjustment of the number of Passive tasks per core often makes a marginal improvement—due to a fractional amount of resource capacity that can be exploited by adding more Passive tasks, we adopt this formula without further refinement on it.

$$N = \begin{cases} \lfloor \frac{1 - CPU^{used}}{CPU^{used} * \frac{numCores}{maxContainers}} \rfloor & \text{if } CPU^{eff} < 0.9 \wedge IO^{wait} \leq T \\ 1 & \text{if } IO^{wait} > T \end{cases} \quad (1)$$

where $Slot^{MAX}$ is the maximum number of allocated slots, T is a threshold configured by the user to determine the characteristic of running tasks, CPU^{eff} and CPU^{used} are the summation of user mode and system mode usage of CPU and the actual usage of CPU, respectively, and IO^{wait} is the average I/O wait.

We enforce a minimum of one Passive task per resource allocation group for this work, although for some applications it may be desirable to allow a minimum of zero Passive tasks (i.e., one Active and two Frozen tasks in a resource allocation group). We suggest that exact amount of resources to allocate to each group of tasks, determined in Hadoop YARN by the amount of resources allocated to each container, will be able to be profitably tuned on a per-cluster basis. This is due to environment specific concerns such as the processing or disk capabilities of the physical nodes used in the cluster. This requires no additional configuration of the Hadoop YARN cluster than what was previously necessary. With one Frozen

Please cite this article in press as: T. Ryan, Y.C. Lee, Multi-Tier Resource Allocation for Data-Intensive Computing, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.03.001

task per resource allocation group, the maximum number of Passive tasks per group is obviously limited to two; in practice we observe that this is sufficient to achieve high resource utilisation. Based on initial testing we configure Hadoop YARN with two CPU cores per resource group. Each resource group consists of three tasks, which are allocated to container tiers such that we have one Active, one Passive, and one Frozen per two CPU cores by default.

4. Evaluation

In this section, we show the efficacy of MTRA with experimental results using both typical MapReduce benchmarks and Facebook workload on an Amazon EC2 cluster. We also include the performance of the implementation of the two-tier resource allocation model [6] in Hadoop YARN (denoted YARN-LRS in the following discussion) in our results to demonstrate the effect of adding the third resource allocation tier on performance.

4.1. Experimental setup

We use the Amazon Elastic Compute Cloud (EC2) as the cluster environment. Our test cluster is made up of 11 m3.2xlarge EC2 instances. m3.2xlarge instances each have 8 vCPUs on Intel Xeon E5-2670 v2 processors, 30 GB of RAM, and SSD based storage. While further information on the underlying physical machines in the EC2 is not available to the public, we follow previous works in choosing the largest instance size in the m3 family as it is likely that these instances each correspond to a physical machine. This is advantageous since it means we are less likely to share a machine with other EC2 users, thereby giving us less interference and more reliable results. Based on the 'rule of thumb' for configuring Hadoop YARN with two containers per CPU core [15], we use YARN configurations with both one and two containers per CPU core as a baseline for comparisons (denoted YARN-1 and YARN-2 respectively).

4.2. MapReduce benchmarks

We first evaluated performance across the canonical MapReduce benchmarks taken from the MR literature [6,16–19]. These benchmarks represent a range of different MapReduce resource usage profiles. A summary of each of the benchmarks is provided in Table 1. Input datasets of 30 GB were used for each benchmark, with the exception of PiEst and TeraSort. PiEst uses 1500 maps with 1 000 000 samples per map and TeraSort uses 10 GB of input data. These input sizes were chosen so as to be sufficiently large to utilise the whole test cluster. To evaluate performance using different schedulers we run experiments using both the Hadoop YARN Capacity Scheduler and FIFO Scheduler. Results are shown in Fig. 2.

Table 1
Overview of MapReduce benchmarks.

Benchmark	Bounding resource	Comments
PiEst	CPU	Estimate Pi using the Quasi-Monte Carlo method
Grep	CPU	Search text data for matches to a regular expression
Word Count	Mixed	Count the number of words in a text data file
Sort	I/O	Sort random input data
TeraSort	I/O	Sort random input data

First discussing the results for the Capacity Scheduler, we see that when using the MTRA approach in YARN (denoted YARN-MTRA), performance is equivalent to or better than the previous best performing configuration for all benchmarks. YARN-MTRA has reduced execution times by 5% on average compared to the best previous YARN configuration, including a performance increase of 18% for the I/O-bound Sort benchmark. YARN-MTRA improves performance by 2% on average compared to YARN-LRS, showing that the three-tier dynamic resource allocation approach is an improvement on the two-tier model. Importantly, YARN-MTRA significantly outperforms YARN-LRS on the Word Count benchmark, which was the worst-performing benchmark for YARN-LRS using the Capacity Scheduler. Performance on the Word Count benchmark is an important indicator of performance for jobs bounded by long-running individual tasks. Next looking at the FIFO Scheduler, we see that when using the MTRA approach in YARN, performance is improved by 4% on average compared to all the configurations, and by up to 13% compared to YARN-2 on several benchmarks.

Fig. 2 shows that when using the FIFO Scheduler, performance using YARN-MTRA is worse than YARN-1 for Word Count, Sort and TeraSort benchmarks. Overall, YARN-MTRA is 2% worse than YARN-1 on average, and up to 6% worse for the Sort benchmark. This is due to the behaviour of the FIFO Scheduler when a job has a small number of concurrent tasks relative to the cluster size. In this situation, the FIFO Scheduler tends to allocate many tasks to the same node. Load imbalance across the cluster is problematic using YARN-MTRA since on the heavily used nodes, some tasks will be allocated to the Passive and Frozen tiers, while they would be allocated to the Active tier if they were located on other nodes. Since Word Count, Sort and TeraSort have a small number of parallel tasks for most of their execution, YARN-1 performs better than YARN-MTRA. However, we suggest that this is not expected to be a problem in real-world systems for two reasons. Firstly, MapReduce applications can be easily modified to request task container locations through the Hadoop YARN API. If it is known that a multi-tier resource management strategy is in use, resources can be requested on different nodes, solving the problem completely. Secondly, in a real-world, multi-user, multiple job environment,

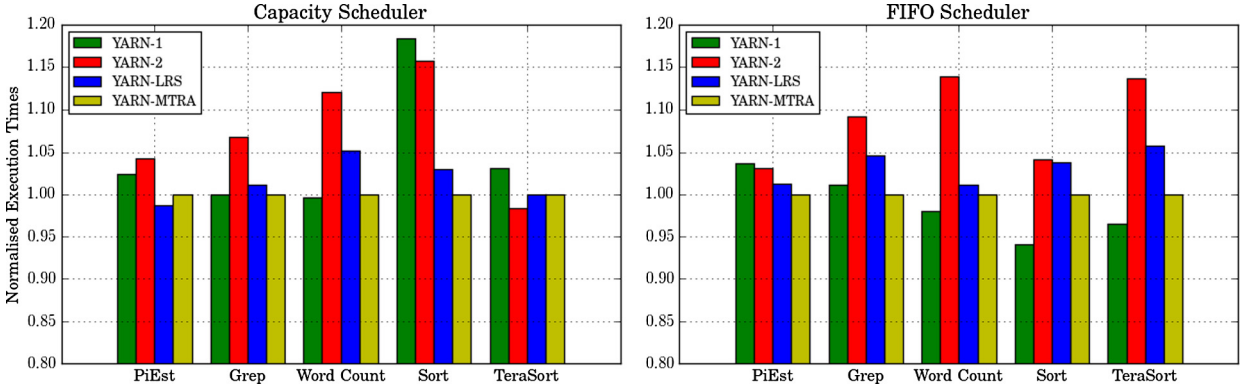

Fig. 2. Performance comparison for MapReduce benchmarks. Results are normalised against YARN-MTRA.

Please cite this article in press as: T. Ryan, Y.C. Lee, Multi-Tier Resource Allocation for Data-Intensive Computing, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.03.001

there are much less likely to be wasted containers since multiple jobs request resources. This means that even though we see decreased performance for some individual jobs, cluster throughput in real-world scenarios will be unaffected.

The fact that for all the benchmarks where the cluster is saturated with tasks YARN-MTRA is within a few percentage points of the otherwise best performing resource allocation technique, and on average improves performance for these benchmarks, shows that the technique is working as intended to optimally adjust resource allocations to individual resource requirements. We do not expect a larger performance gain here since the advantage of MTRA lies mainly in equalling the best static configuration. The results show that MTRA has achieved this, as we are not significantly worse than any other resource management approach on any benchmark, even for cases where MTRA is at a disadvantage due to the behaviour of the scheduler or application.

4.3. Facebook workload

Typical workloads on real-world production clusters have a range of jobs with multiple resource usage profiles running concurrently. As such, the individual benchmarks themselves are only of limited value in assessing the performance of a system in a real-world environment. To evaluate the performance in a multi-job, mixed-workload environment, a workload model was developed based on reported workloads in a Facebook production cluster [20,6]. The model uses the reported distribution of job input sizes, job resource usage characteristics, and the amount of time between job submissions to simulate the Facebook production environment. Based on the previously reported figures, 25 jobs were used in the workload model. These were submitted at intervals of 14 seconds, for a submission schedule of 350 seconds in length. The input sizes used were in the same ratio as those used in the Facebook workload, scaled down relative to the size of our test cluster. We used 64 MB of input data per Map task for each job. Table 2 shows a breakdown of the individual jobs and input sizes for the workload model.

For our analysis we focus on the execution times of individual jobs, as this is the most meaningful metric to end users. As shown in Fig. 3, YARN-MTRA improves job execution time by 9% compared to both YARN-2 and YARN-LRS, and 13% compared to YARN-1, resulting in a 10% performance improvement on average. Analysis of individual job execution time shows that this improvement is not dependant on job duration, with the same relative performance

Table 2
Facebook workload model application breakdown. Percentage of total jobs (% Total jobs) is based on the original trace.

# Tasks/job	% Total jobs	# Benchmarks	Total input data size for each job (MB)
1–2	54%	8 Word Count	64
		6 TeraSort	128
3–20	16%	2 Sort	512
		2 Word Count	1024
21–150	12%	1 PiEst	5120
		2 TeraSort	7680
151–300	6%	1 Word Count	15 360
301–500	4%	1 PiEst	25 600
500+	7%	1 Sort	33 280
		1 Grep	40 960

across jobs with both long and short execution times. For individual jobs we observe reduction in execution time of up to 56% in I/O bound benchmarks.

These results show two significant things. Firstly, they demonstrate the importance of the ability of the resource management strategy to deal with heterogeneous application resource requirements; while YARN-LRS performed better than both YARN-1 and YARN-2 on average in the individual benchmarks, it performed worse on the Word Count application. Analysis of individual job execution times in the workload shows that YARN-LRS performed worse for the several Word Count jobs in the workload, resulting in the overall decreased performance we observe. Secondly, these results confirm our earlier hypothesis that the instances where YARN-MTRA performed worse (due to slot under-saturation when jobs only launch a small amount of parallel tasks) are not a significant problem in a multi-job environment.

Overall performance improvements for individual job execution times are slightly larger than they were for the individual MapReduce benchmarks due to the potential for greater performance improvement if tasks with complementary resource usage profiles are co-located. For example, if an I/O bound Active task and a CPU bound Passive task are placed together, we see much greater performance improvements since the CPU bound task receives more CPU resources than it would if the Active task was also CPU bound. There is very little opportunity for this complementary co-location to occur in a single MapReduce job since individual tasks within MapReduce applications tend to have very similar, if not identical,

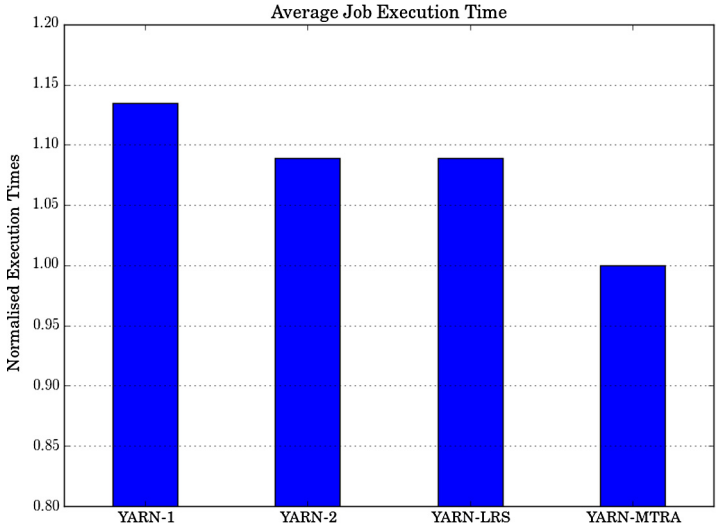

Fig. 3. Average execution time of jobs in the workload. Results are normalised against YARN-MTRA.

Please cite this article in press as: T. Ryan, Y.C. Lee, Multi-Tier Resource Allocation for Data-Intensive Computing, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.03.001

resource usage profiles [6]. These results show that MTRA can significantly improve performance in a real-world MapReduce cluster by accounting for heterogeneity of resource requirements.

5. Related work

While there are many resource management strategies that focus on efficient resource management at the task level in distributed systems, nearly all of this work implements resource management strategies at the level of the cluster scheduler. Additionally, most schedulers are optimised for performance in specific environments and use-cases; the MTRA resource management strategy is scheduler-independent, and able to be applied to any distributed environment where applications are made up of many small tasks. We have stressed the usefulness of MTRA in mixed workload environments due to task resource requirement heterogeneity. This also applies to environments such as Mesos and Omega, which are likely to have much greater heterogeneity due to diversity of not only jobs but of frameworks and schedulers [21,22]. Several approaches which attempt to increase resource utilisation do this by virtual machine (VM) manipulation [23]; while there are parallels with MTRA, we focus on finer-grain resource allocation. Additionally, the generality of the MTRA technique suggests that it could be useful at levels other than just the task level.

Resource management approaches based on co-location of tasks with certain task resource usage profiles is complementary to our work [24]. Typically these strategies attempt to co-locate tasks which require different resources in order to reduce contention; however, these approaches require accurate profiling of jobs' usage requirements. This is not a trivial task, and adds significant complexity to the resource management process. Most of these systems are based on comparing incoming tasks to those that have previously been seen, limiting the usefulness of the system for new applications [18,17,25]. Despite these concerns, we have observed when running the workload model that co-location of tasks with complementary resource usage profiles in MTRA can result in very large performance increases. With this in mind, the application of MTRA to a system which co-locates complimentary tasks could be highly advantageous.

There also have been some works (e.g., [26,27]) taking into account multiple dimensions of resource allocation in clouds. The work in [26] addresses the problem of allocating resources to multi-tier cloud applications for the maximisation of profit. Although it considers three dimensions of resource allocation, processing, memory requirement and communication resources, its granularity of resource allocation is coarser than that in MTRA. In the meantime, the multi-dimensionality in resource allocation referred by Jrad et al. in [27] is higher level than that in MTRA in that the work in [27] considers cost and data locality as multiple objectives for running scientific workflows (e.g., Montage astronomical mosaic engine [3,4]) across multiple clouds.

6. Conclusion

In this paper, we have presented a novel resource allocation technique (MTRA) that dynamically adjusts resource allocations to individual tasks. By introducing a third resource allocation tier, MTRA adjusts resource allocations in a fine-grained manner based on resource usage levels on each cluster node. It is scheduler-independent, resulting in increased scalability, and also allows finer-grained control than would be possible with a scheduler based solution. In our evaluation, we have implemented MTRA in Hadoop YARN. We have observed performance improvements of up to 18% compared to Hadoop YARN for individual MapReduce benchmarks. For a multi-job workload model based on a Facebook production cluster workload, we have shown that MTRA reduces individual job execution times by 10% on average and up to 56% for individual applications. We conclude that our novel dynamic resource allocation technique succeeds in meeting the need for a general-purpose scheduling technique which can account for application heterogeneity.

Acknowledgement

The authors would like to thank Peng Lu for his initial contribution and helpful discussions on the actual implementation in Classic Hadoop. Dr. Young Choon Lee would like to acknowledge the support of the Australian Research Council Discovery Early Career Researcher Award Grant DE140101628.

References

[1] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, OSDI '04, USENIX Association, 2004, pp. 137–150.
[2] G. Malewicz, M.H. Austern, A.J. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, ACM, New York, NY, USA, 2010, pp. 135–146.
[3] J.C. Jacob, D.S. Katz, Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking, Int. J. Comput. Sci. Eng. 4 (2) (2009) 73–87.
[4] Montage: an astronomical image mosaic engine [online; accessed November 2, 2014], http://montage.ipac.caltech.edu/, 2014.
[5] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, Apache Hadoop YARN: yet another resource negotiator, in: Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13, ACM, New York, NY, USA, 2013, pp. 5:1–5:16.
[6] P. Lu, Y.C. Lee, V. Gramoli, L.M. Leslie, A.Y. Zomaya, Local resource shaper for MapReduce, in: Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Singapore, 2014, pp. 483–490.
[7] T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft, 2009.
[8] Amazon, Amazon elastic compute cloud [online; accessed November 2, 2014], http://aws.amazon.com/ec2/, 2014.
[9] Google, Google compute engine [online; accessed November 2, 2014], https://cloud.google.com/compute/, 2014.
[10] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[11] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for QoS-aware clouds, in: Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, ACM, 2010, pp. 237–250.
[12] G. Jung, M.A. Hiltunen, K.R. Joshi, R.D. Schlichting, C. Pu, Mistral: dynamically managing power, performance, and adaptation cost in cloud infrastructures, in: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, ICDCS '10, IEEE Computer Society, 2010, pp. 62–73.
[13] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, vol. 2, NSDI'05, USENIX Association, 2005, pp. 273–286.
[14] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: fair allocation of multiple resource types, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, USENIX Association, Berkeley, CA, USA, 2011, pp. 323–336.
[15] Hortonworks, How to plan and configure YARN in Hadoop 2.0 [online; accessed November 2, 2014], http://hortonworks.com/blog/how-to-plan-and-configure-yarn-in-hdp-2-0/, 2014.
[16] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, E. Harris, Reining in the outliers in map-reduce clusters using Mantri, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 1–16.
[17] P. Lu, Y.C. Lee, C. Wang, B.B. Zhou, J. Chen, A. Zomaya, Workload characteristic oriented scheduler for MapReduce, in: Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012, pp. 156–163.
[18] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, E. Ayguadé, Resource-aware adaptive scheduling for MapReduce clusters, in: Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware, Middleware '11, Springer, Berlin, Heidelberg, 2011, pp. 187–207.
[19] A. Verma, L. Cherkasova, R. Campbell, Resource provisioning framework for MapReduce jobs with performance goals, in: F. Kon, A.-M. Kermarrec (Eds.),

Middleware 2011, in: Lect. Notes Comput. Sci., vol. 7049, Springer, Berlin, Heidelberg, 2011, pp. 165–186.

[20] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, in: Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, ACM, New York, NY, USA, 2010, pp. 265–278.

[21] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: a platform for fine-grained resource sharing in the data center, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, 2011, pp. 295–308.

[22] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, J. Wilkes, Omega: flexible, scalable schedulers for large compute clusters, in: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, ACM, New York, NY, USA, 2013, pp. 351–364.

[23] J. Park, D. Lee, B. Kim, J. Huh, S. Maeng, Locality-aware dynamic VM reconfiguration on MapReduce clouds, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, HPDC '12, ACM, New York, NY, USA, 2012, pp. 27–36.

[24] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A.R. Butt, N. Fuller, MRONLINE: MapReduce online performance tuning, in: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC '14, ACM, New York, NY, USA, 2014, pp. 165–176.

[25] C. Delimitrou, C. Kozyrakis, QoS-aware scheduling in heterogeneous datacenters with paragon, ACM Trans. Comput. Syst. 31 (4) (2013) 12:1–12:34.

[26] H. Goudarzi, M. Pedram, Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems, in: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD '11, IEEE Computer Society, 2011, pp. 324–331.

[27] F. Jrad, J. Tao, I. Brandic, A. Streit, Multi-dimensional resource allocation for data-intensive large-scale cloud applications, in: Proceedings of the 4th International Conference on Cloud Computing and Services Science, CLOSER '14, 2014.

Please cite this article in press as: T. Ryan, Y.C. Lee, Multi-Tier Resource Allocation for Data-Intensive Computing, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.03.001