H-0324 (HAI1606-001) 1 June 2016
Computer Sciences

IBM Research Report

Megos: Enterprise Resource Management in Mesos Clusters

Abed Abu-Dbai
IBM Research – Haifa

Khalid Ahmed
IBM Platform Computing, Toronto, CA

David Breitgand
IBM Research – Haifa

Gidon Gershinsky
IBM Research – Haifa

Alex Glikson
IBM Research – Haifa

Research Division
Almaden – Austin – Beijing – Brazil – Cambridge – Dublin – Haifa – India – Kenya – Melbourne – T.J. Watson – Tokyo – Zurich

Megos: Enterprise Resource Management in Mesos Clusters

Abed Abu-Dbai
IBM Research – Haifa

Khalid Ahmed
IBM Platform Computing, Toronto, CA

David Breitgand
IBM Research – Haifa

Gidon Gershinsky
IBM Research – Haifa

Alex Glikson
IBM Research – Haifa

Abstract

Enterprise data centers increasingly adopt a cloud-like architecture that enables the execution of multiple workloads on a shared pool of resources, reduces the data center footprint and drives down the costs. The Apache Mesos project is emerging as a leading open source resource management technology for server clusters. However, the default Mesos allocation mechanism lacks a number of policy and tenancy capabilities, important in enterprise deployments. We have investigated integration of Mesos with the IBM EGO (enterprise grid orchestrator) technology which underpins various high performance computing, analytics and big data clusters in a variety of industry verticals including financial services, life sciences, manufacturing and electronics. We have designed and implemented an experimental integration prototype, and have tested it with SparkBench workloads. We demonstrate how Mesos can be enriched with new resource policy capabilities, required for managing enterprise data centers.

1 Introduction

Enterprise workloads become increasingly diverse. Even within the same class of applications, such as Big Data analytics, workload types range from pure batch to interactive [5, 10]. Therefore, an ability to use the same resource pool for executing heterogenous workloads, owned by different parties in the enterprise (e.g., departments, groups, and teams), becomes critically important for cost-efficiency.

To deal with this challenge, a number of cluster resource managers have appeared over the last few years, aimed at providing a uniform technology-neutral resource representation and management substrate allowing to harmonize execution of long run services, batch and interactive jobs on the same infrastructure comprising physical resources, virtual ones or combination thereof. Examples include YARN [4], Borg [15], Omega [13], Mesos [8], and EGO [9].

Being driven by similar incentives, these resource management frameworks share many features. However, they also differ considerably on the richness of their respective resource models, supported resource management policies, maturity levels and ownership (proprietary *e.g.*, Borg, Omega, EGO vs open source *e.g.*, Apache YARN and Mesos). Consequently, the choice of a particular resource manager is influenced by multiple considerations.

Mesos recently exploded as an extremely popular Apache project with a number of large organisations successfully using it for production [1, 12, 14, 7]. Mesos offers simple yet powerful and flexible APIs, highly available and fault tolerant architecture, scalability to large clusters, isolation between tasks using Linux containers, multi-dimensional resource scheduling, ability to allocate shares of the cluster to *roles* representing users or user groups, and a clear separation of concerns between the applications (termed frameworks) and the "cluster kernel", which is Mesos. The resource scheduler of Mesos supports a generalization of max-min fairness, termed Dominant Resource Fairness (DRF) [6] scheduling discipline, which allows to harmonize execution of heterogeneous workloads (in terms of resource demand) by maximizing the share of any resource allocated to a specific framework.

There are a number of areas where Mesos can be improved to meet enterprise customer requirements. In this paper, we present a work in progress focusing on policies that are present in enterprise-grade resource managers, but are missing in Mesos. We demonstrate how Mesos can be enriched with IBM Platform EGO features [9], such as

- Capturing of the hierarchical structure of an enterprise (organisations, departments, groups, teams, users) by defining the corresponding resource *con-*

sumer tree;

- A fine grained resource plan allowing to define resource *share ratio, ownership* and *lending/borrowing* policies for each resource consumer;
- A rich set of resource management policies making use of the hierarchical resource consumer model and providing fairness and isolation to the members of hierarchy including an important ability to dynamically change the allocations (*time-based policy*);
- A Web-based GUI providing a centralized console through which the whole cluster is observed and managed. In particular, the cluster-wide resource management policies are applied through this GUI.

Our specific contributions are as follows. First, we describe an experimental architecture for loose coupling between Mesos and EGO resource management mechanisms. Second, we implement the proposed architecture, called Megos, and evaluate it using Spark based workloads. Third, we demonstrate through experimentation that the enterprise policies (e.g. hierarchical shares management and time-dependent share re-allocation), currently missing in Mesos, can be efficiently applied to the existing Mesos frameworks, by linking the Mesos master to the EGO policy mechanism.

The main purpose of this work is to demonstrate feasibility and benefits of EGO-style policies for Mesos to achieve enterprise-grade cluster resource management.

2 Related Work

Apache YARN [4] provides the functionality of a resource manager and a resource scheduler. These functionalities are separated into different daemons: a global Resource Manager (RM) and per application Application Manager (AM). Application frameworks negotiate resource allocations with the global RM. In that respect, it is similar to EGO [9]. YARN supports scheduling with pluggable scheduling policies. The policies provided off-the-shelf include Fair Scheduler and CapacityScheduler. YARN is an integral part of Hadoop ecosystem. As such, YARN is optimized for running MapReduce applications in a shared, multi-tenant cluster while maximizing the throughput and the utilization of the cluster. The primary abstraction provided by YARN scheduling is a *queue*. To achieve control and predictability of resource sharing, the YARN supports hierarchical queues to ensure resources sharing among the sub-queues of an organization before other queues are allowed to use free resources. This way, affinity for sharing free resources among applications of a given organization is provided. While this feature is similar to EGO, it should be noted that (a) YARN is not intended as a general-purpose resource manager and (b) the policy model of YARN is less expressive than that of EGO. In particular, there are no concepts of ownership, lending and borrowing in YARN. Also, it does not support time-based policies.

Omega [13] is a proprietary data center manager that employs an optimistic concurrency control by offering each application framework a local copy of the cluster state. Omega does not support fairness. Rather, the high priority jobs (service ones) can preempt low priority ones (batch MapReduce ones). Omega is not a publically released technology, making an explicit comparing Megos difficult.

Production experience with Borg was recently reported by Google [15]. Borg shares many assumptions with Omega, because it serves the same Google workloads. Borg copes with extreme heterogeneity and reportedly scales to tens of thousands of nodes. Again, direct comparison to Megos is difficult as Borg is proprietary.

3 EGO Enterprise Policies

EGO manages resource allocation via resource plans, configured by cluster administrator using the Platform Management Console (PMC) web interface. Resource plans are comprised of a *consumer tree* and *resource groups*, mapped to the tree.

The consumer tree reflects the organizational structure (root/department/group/consumer), with the leaf nodes being the consumer entities that actually run the jobs and service applications. The tree outlines organizational relationships among consumers, while the plan specifies a policy of resource allocation. The choice of consumers and their hierarchy should reflect long-term business goals. The resource groups are non-overlapping sets of cluster resources (hosts), selected according to some criteria.

The resource plan defines how cluster resources are allocated among consumers. The plan takes into account the differences between consumers and their needs, resource properties, and various other policies concerning consumer rank and the allocation of resources. Resources are allocated to a consumer through *ownership, borrowing, or sharing*. Ownership refers to the guaranteed allocation of a minimum number of resources to a consumer. Borrowing is a temporary allocation of owned resources from a lending consumer to a consumer with an unsatisfied demand. Sharing refers to the temporary allocation of unowned resources from a *share pool* to a consumer with an unsatisfied demand.

When a consumer experiences demand, EGO considers and allocates resources in the following order (by

default): 1. Idle resources already owned by the consumer. 2. Idle, unowned resources from the share pool. 3. Idle resources owned by other consumers that are configured for lending (borrowed resources). 4. Resources owned by the consumer but currently lent-out to other consumers (reclaimed resources to owner). 5. Unowned resources from the share-pool but currently in use by consumers with a smaller share-ratio (reclaimed resources to share-pool).

It is possible to change the default resource allocation policy so that owned resources get reclaimed by consumers before they are borrowed or allocated from elsewhere. Also, allocation policy can be adjusted so that resources are never reclaimed by the share pool, but are only returned when the borrowing client releases them.

The resource plans can change according to time of day (master host timezone), using a *time-based* configuration option. This allows to have different e.g., daytime and night-time resource allocation policies, or to modify the policies according to the business hours in different branches of the organization (e.g., Europe, US, Asia-Pacific).

The resource allocation can be performed using either Slot scheduling or Multi-dimensional scheduling (MDS)[1]. With Slot scheduling, each host in a resource group is logically split into a number of slots that become the units of resource allocation and runtime management. The number of slots per host can be defined by the cluster administrator, or automatically by an expression, taking into account the host CPU and memory parameters.

4 Design and Implementation

Mesos uses a bottom-up resource offer mechanism, where the free host resources are detected by an agent (slave), and offered to Mesos applications (frameworks) via the master node. EGO uses a top-down approach where the applications request specific amount of resources. There is no straightforward mapping of the two mechanisms, but its possible to integrate these resource managers in a number of ways, with a final result being application of EGO policies to Mesos framework management. In this paper, we describe an experimental loosely coupled integration approach that allows to demonstrate EGO policies in Mesos with minimal code changes. It is a work in progress, focused on a proof-of-concept demonstration; later designs might use a different integration approach.

[1] With MDS, the allocation units are simply the hosts (absolute number of hosts, assigned to a consumer, within its resource group) or a percentage of each host in a resource group (specified by resource metrics, such as CPU, memory, etc), assigned to a consumer.

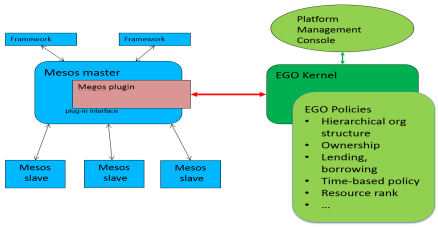

Figure 1: Megos architecture

The loosely coupled design requires no changes in Mesos frameworks or agents (slaves), and it works with the existing EGO technology. The integration is performed as a custom Mesos allocator, using the standard plug-in API in Mesos master [2]. We take the existing Mesos DRF allocator source code as a basis for the implementation.

Megos plug-in uses EGO C APIs to connect to EGO Kernel process, deployed on a different host, or co-located with the Mesos master. The EGO Kernel is configured with a hierarchical resource management policy, created via the PMC (Platform Management Console) GUI, as shown in the example below (Figure 2). The figure shows a resource group called MesosGroup2. The consumer tree is comprised of a parent (Mesos), two departments (DeptA and DeptB) and a number of leaf consumers - all sharing the same cluster of hosts (MesosGroup2). The resource plan specifies the policy parameters - ownership, lend limit, share ratio - at different layers in the hierarchy. Here we use slot-based scheduling for simplicity. The host slots are summed up into cluster slots, and can be distributed by an administrator as cluster-wide resource ownership to different departments and consumer leaves in the organization.

We map each EGO leaf consumer to a Mesos *role* (full path, eg "/Mesos/DeptB/ConsB1" is a role). Upon Mesos master start-up, the Megos allocation plug-in module loads and connects to EGO Kernel as an EGO client, using pre-configured credentials. Then, in a loop over all defined roles, Megos executes EGO allocation requests for every consumer leaf, and collects allocation or reclaim responses. EGO sends a reclaim message when an under-allocated consumer has a demand for resources currently held by an over-allocated consumer. In a few seconds, Megos has a full allocation map - the number of slots, allocated by EGO to each consumer/role, according to the current EGO hierarchical policy. We simply use this number as a weight in the DRF mechanism, since it reflects the proportional share of cluster resources for the Mesos role represented as an EGO consumer leaf. The weights are not static - we perform the EGO allocation cycle once in a minute. This is

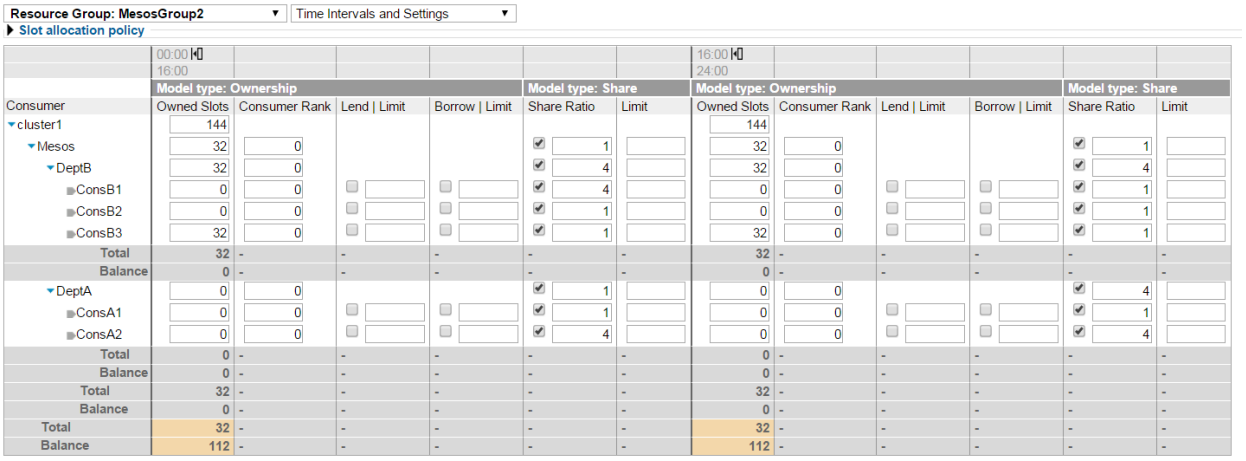

Figure 2: Configuring resource plan in EGO GUI

done only for the active Mesos roles - the ones with registered frameworks. If a role doesn't have a framework yet, Megos won't request EGO slots for it, but will set its weight to a minimal value (1), so the role still can receive some resource offers if a new framework is registered to it later. In the next allocation cycle, this role will be configured with its proper weight, since it became active in the meantime.

The EGO ownership is explicitly mapped to Mesos static reservations. This is a simple PoC implementation based on the available Mesos features at the time, and can be replaced with the new Mesos Quota mechanism released in Mesos v0.27.

Once the Mesos DRF weights and static reservations are configured according to the EGO policies, the subsequent resource management process is performed by the standard Mesos mechanism - the resource offers are generated by the agents (slaves), and distributed to the frameworks according to reservation and DRF parameters. The only runtime intervention by EGO is the periodic re-configuration of DRF weights, as described above, to handle changes in active roles, and in the EGO policies (for example, the time-based policies).

5 Experiments

5.1 Testbed

Our testbed comprises a cluster of 11 machines. Nine machines have been used as compute nodes and the remaining two have been used to run masters of Mesos, Hadoop, and EGO. The Mesos master and the Hadoop master (manages the HDFS file system) run together on a Ubuntu 14.04 machine. The EGO master runs in a separate Red Hat machine. The remaining 9 machines are Ubuntu 14.04 that run the EGO LIM daemons, the Mesos slaves, and the data-nodes of Hadoop-HDFS. Every compute host has 4 CPU's and 8GB of physical memory.

We use Mesos verion 0.23.0, Hadoop version 2.4 and the compatible Spark version 1.5 . The experiments are based on SparkBench benchmark suite [11], that covers a number of application categories. In this work, we employ Spark Logistic Regression (LR) sample job as a workload for the tests described in the following sections.

5.2 Time-based policy

Time-based policy is a feature that allows to apply changes in resource sharing policy based on the time of day. In this experiment we divided the day into two intervals: 00:00-16:00 and 16:00-24:00. In the first interval we configured the share ratio (weight) to be 1 for the consumer "/Mesos/DeptB/ConsB1" and 4 for "/Mesos/DeptB/ConsB2". In the second interval (starting at 16:00) we switched the share ratio and the weight of consumer "/Mesos/DeptB/ConsB1" became 4 while the weight of consumer "/Mesos/DeptB/ConsB2" became 1. The test is 4 hours long, we started it at 14:00 so the policy's settings would be switched at the middle of the run. F1 and F2 are the frameworks registered to consumers "/Mesos/DeptB/ConsB1" and "/Mesos/DeptB/ConsB2" respectively. The frameworks run identical workloads of LR iterations in a loop. We use 8 sample points every 30 minutes and measure the job rate (iterations/hour) of each framework in every sample point. Figure 3 shows the job rate of F1, F2, and the total rate of F1+F2 in Megos. In the middle of the run (2 hours from start), the job rate of F1 starts increasing while F2 diminishes as a result of the automatic policy switch in Megos.

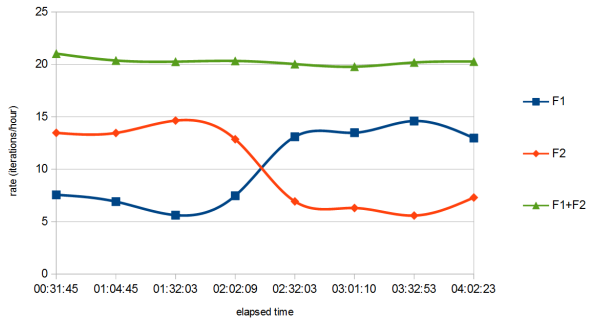

Figure 3: Job rates with time-based policy

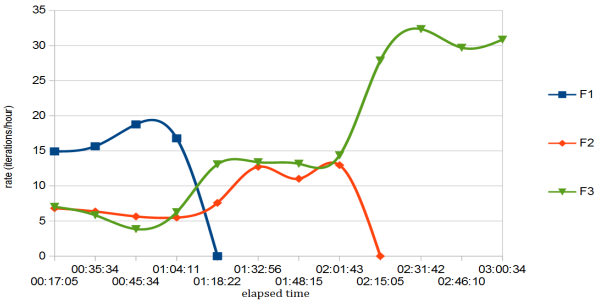

Figure 4: Mesos resource sharing behavior

5.3 Hierarchical resource sharing

Department considerations are absent in the standard Mesos. To illustrate this, we run three identical Spark frameworks, where each framework is running LR iterations as a workload and uses three different consumers from two departments. Frameworks F1 and F2 are registered to consumers ”ConsA1” (weight 4) and ”ConsA2” (weight 1) in the department ”/Mesos/DeptA” (department weight 4) while F3 is registered to a consumer ”ConsB1” (weight 4) from the department ”/Mesos/DeptB” (department weight 1).

Since the standard Mesos doesn't support department hierarchy, we configure three roles: ”/Mesos/DeptA/ConsA1”, ”/Mesos/DeptA/ConsA2”, and ”/Mesos/DeptB/ConsB1” with the accumulated weights of 16, 4, and 4 respectively. Besides, we use 12 sample points every 15 minutes and a smaller regression data than in the previous tests (leading to higher job-rates than before). In the beginning of the test, we run all frameworks in parallel (Figure 4). The first framework (F1) runs for one hour and stops. Its resources are freed - and divided between the other two frameworks without department considerations. That is why the job-rates of both F2 and F3 increase equally. Later, when F2 finishes, its resources are allocated to F3.

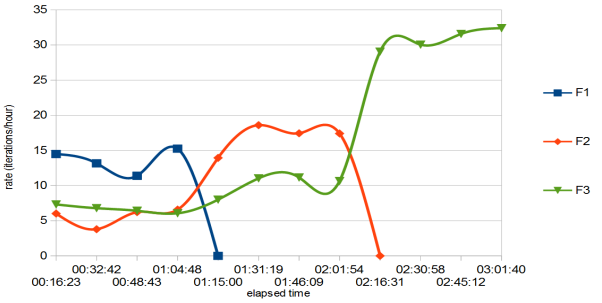

Figure 5: Megos resource sharing behavior

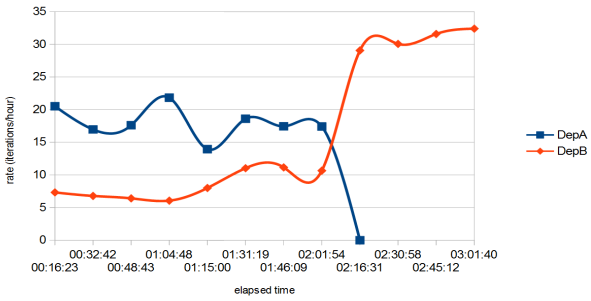

Figure 6: Megos: Total job rates of the departments

Next, we use Megos to run the same experiment. We show in Figure 5 and Figure 6 that freed resources remain in the same department. When framework F1 stops at 01:00:00, its resources are given to its sibling in DeptA - the framework F2, leading to significant increase in F2 job rate. The rate of F3, belonging to another department, doesnt change much. The total rate of DeptB stays lower than DeptA after the end of F1 run at 01:00:00.

6 Conclusions and Future Work

This paper presents Megos, a resource management system that extends the Apache Mesos capabilities by integrating with the IBM Platform EGO policy mechanism.

We demonstrate feasibility and usefulness of applying such policies to the existing Mesos frameworks. Apache Mesos can be enhanced by either coupling it with an external enterprise policy mechanism, or by direct contribution of the relevant features to the Mesos code, or by a combination thereof. For example, the concept of resource lending and reclaim is already being integrated into core Mesos [3]. This and other advanced features will expand the Mesos management model and allow it to address a wider set of requirements in enterprise data centers.

References

[1] ANISZCZYK, C. Apache Mesos at Twitter. Texas LinuxFest, 2014.

[2] APACHE MESOS PROJECT. Mesos Allocation Modules. `http://mesos.apache.org/documentation/latest/allocation-module/`, 2016.

[3] APACHE MESOS PROJECT. Mesos Optimistic Offers. `https://docs.google.com/document/d/1RGrkDNnfyjpOQVxk_kUFJCalNMqnFlzaMRww7j7HSKU/edit#heading=h.qa9kakriyq0d`, 2016.

[4] APACHE SOFTWARE FOUNDATION. Apache Hadoop YARN. `https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`, 2016.

[5] CHEN, Y., ALSPAUGH, S., AND KATZ, R. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment 5*, 12 (2012), 1802–1813.

[6] GHODSI, A., ZAHARIA, M., HINDMAN, B., KONWINSKI, A., SHENKER, S., AND STOICA, I. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI* (2011), vol. 11, pp. 24–24.

[7] HARRIS, D. Scaling Mesos at Apple, Bloomberg, Netflix and more. Mesosphere Technical Blog, Aug 2015.

[8] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R. H., SHENKER, S., AND STOICA, I. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI* (2011), vol. 11, pp. 22–22.

[9] IBM PLATFORM COMPUTING. An Introduction to EGO: an enterprise ready resource manager for all workloads. `https://www.ibm.com/developerworks/community/wikis/form/anonymous/api/wiki/250c325e-a324-491c-9006-999ad6e1df9f/page/ab943ccd-542a-4c53-bb36-b757d16466bf/attachment/6a26aeed-214f-44b2-acce-91adcdfed13e/media/EGO_whitepaper_v1.pdf`, 2015.

[10] KAMBATLA, K., KOLLIAS, G., KUMAR, V., AND GRAMA, A. Trends in big data analytics. *Journal of Parallel and Distributed Computing 74*, 7 (2014), 2561–2573.

[11] LI, M., TAN, J., WANG, Y., ZHANG, L., AND SALAPURA, V. Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark. In *Proceedings of the 12th ACM International Conference on Computing Frontiers* (2015), ACM, p. 53.

[12] MATTHEWS, B. How Airbnb Simplifies with Mesos. `https://engineering.twitter.com/university/videos/how-airbnb-simplifies-with-mesos`, 2014.

[13] SCHWARZKOPF, M., KONWINSKI, A., ABD-EL-MALEK, M., AND WILKES, J. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013), ACM, pp. 351–364.

[14] THE EBAY PAAS TEAM. Delivering eBay's CI Solution with Apache Mesos. eBay Technical Blog, Apr 2014.

[15] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 18.