ADARM: An Application-Driven Adaptive Resource Management Framework for Data Centers

Min Luo
Shannon Lab
Huawei
Santa Clara, CA, USA
min.ch.luo@huawei.com

Li Li
Shannon Lab
Huawei
Bridgewater, NJ, USA
li.nj.li@huawei.com

Wu Chou
Enterprise Network
Huawei
Bridgewater, NJ, USA
wu.chou@huawei.com

***Abstract* — ADARM is a framework for application-driven and adaptive resource allocation to enable large scale service-oriented cloud DCs. It builds and utilizes real-time application profiles and models to understand application requirements and performance characteristics. In addition, service-orientation enables ADARM to use uniform and flexible REST services to abstract, organize, monitor and control the fine-grained cloud resources, and it separates the resource control logic from the datacenter infrastructure. The combination of application-driven and service-orientation enables ADARM to incorporate various adaptive optimization techniques to optimize resource allocations for large scale distributed DCs. This paper discuses the design rationales behind ADARM, and describes its architecture and main benefits. Prototypical validations on several core components in ADARM are presented to demonstrate the potential business value and the feasibility of such a system framework.**

Keywords- Software defined data centers; application driven; adaptive resource management; DC analytics; unified resource allocation and optimization.

I. INTRODUCTION

With the rapid advancements of cloud computing and mobile internet technology, it is ever so critical to elevate the capability for enterprise data centers (DCs). Current DCs suffer from low resource utilization, high cost and complexity for operations and management, resulting in significant waste of human resources and energy. While business requirements demand doing more with less, how to make IT infrastructure optimal and adaptive is, therefore, critical. It is inevitable and becomes a core issue to take advantage of the increased computing power and storage resources, and effectively integrate with business aware, more controllable and manageable networking technologies, especially leveraging the software defined infrastructure.

To make DCs flexible, virtualization technologies are applied. Although virtualization helps improve hardware utilization, it is still a challenge to make use of the over-fractionalized resources in DCs. In addition, when virtualization is applied, it can incur significant overhead. As such, in current DC architecture, logical pooling of resources is still restricted by the physical coupling of in-rack hardware devices. Thus, it limits the resource utilization, making it difficult to support big data applications in an effective and economical manner. One of the key steps to address low resource utilization problem is to introduce resource disaggregation, i.e., decoupling processor, memory, and I/O from its original arrangements and organizing these resources into shared pools. Based on disaggregation, on-demand resource allocation and flexible run-time application deployment can be realized with optimized resource utilization and reducing the Total Cost of Operation (TCO) of the infrastructure.

However, conventional DCs only provide limited dynamic management for application deployment, configuration and run-time resource allocation. When scaling is needed in large-scale DCs, lots of complex operations still need to be completed manually. Given the complexity and magnitude of such DCs, it is a huge challenge to maximize the resource utilization and processing power with minimal costs. With tens of thousands or even hundreds of thousands of servers with each server having multiple CPUs, addressable and controllable memory blocks, and thousands or even millions of applications from different tenants, how to manage this large distributed system with uninterrupted, QoS/LSA guaranteed business services will be a huge challenge.

To avoid complex manual re-structuring and re-configuration, intelligent self-management with higher level of automation is needed in DCs. Furthermore, to speed up the application deployment, software defined approaches to monitor and allocate resources with higher flexibility and adaptability is needed.

The management of such an intrinsically distributed system needs to be able to effectively plan, schedule and control large volumes of those resources, so that they all could work collaboratively towards an optimized solution for all resources and supported applications. Therefore, it is highly desired to quickly deploy and operate new businesses, uncover and restore system faults from all components, and eventually achieve large scale and reliable operations management for DCs.

In this paper, we present ADARM, a framework designed for application-driven and adaptive resource management for large scale cloud DCs. It utilizes business

and IT operations modeling at proper abstraction levels. Through profiling, feature extraction and learning, it connects hidden causative factors to performance metrics, and it uses networking connections to optimally configure and deploy resources dynamically in response to workload changes, to achieve resource usage maximization with minimized cost and management complexity. It also adaptively enforces QoS/SLA requirements from tenants or their individual applications. By combining traditional statistical analysis with multi-dimensional modeling and Big Data processing, and utilizing the adaptive optimization techniques, it can enable the near real-time application of ADARM even for large scale distributed DCs.

The rest of this paper is organized as follows. Section II reviews some key characteristics of the data centers which motivate the design of ADARM. Section III describes the ADARM framework, its core components and processes. Section IV discusses the main benefits of ADARM. Section V presents the prototypical validation on several core components. Finally, Section VI presents a summary with some future research directions.

II. KEY CHARACTERISTICS OF CLOUD DATA CENTERS

Data centers (DCs) are expected to meet many challenging requirements, especially PB/s-level data processing capability that can ensure aggregated high-performance and also high-throughput computing [1], with low latency storage and flexible networking, while the business requirements and operation environments continue to change and evolve, as illustrated in Figure 1.

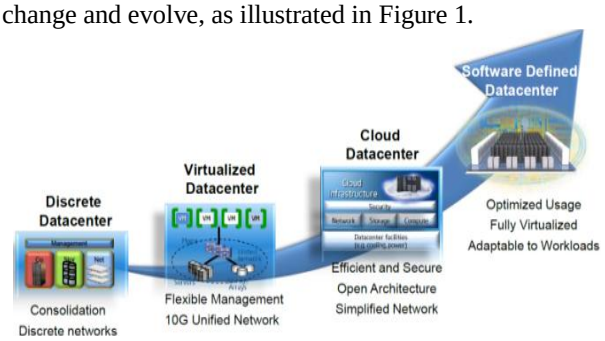

Figure 1: Trend of datacenter architectures

Moreover, data centers/clouds should aim to address the following issues and challenges:

- Application-driven adaptive resource management and allocation optimization.
- Heterogeneous and dependent resources with heterogeneous workloads.
- Automation of resource provisioning and self-organization.
- Real-time monitoring, maintenance and optimized allocation of resources.

To support big data applications, DCs should be enabled with the following features:

Big-Data-Oriented: Different from conventional computing-centric DCs, data-centric could be one of the key design features for DCs. Big data analysis based applications have highly varied characteristics, based on which it should provide optimized mechanisms for rapid transmission, highly concurrent processing of massive data, and also for application-diversified accelerations.

Adaptation for application/task variation: Data-oriented applications have different needs for resource usage priority. To meet such demands with high adaptability and efficiency, disaggregation of all available resources/devices can become a key step to eliminate the tight coupling and enforce the "on demand" satisfaction of resources requirement in a concerted fashion. Such techniques can enable flexible run-time configuration on resource allocation to meet the varied resource demand for different applications.

Intelligent Management: High density deployment and overly utilized virtualization technology give rise to ever-increasing challenge in resource management. Over provisioned network bandwidth not only can make the overall bandwidth utilization low, but could also lead to application (and eventually business) failures or unacceptable delays at critical times. For example, the virtual networking configuration and provisioning is notoriously complex, manual/script intensive, and therefore, it is not only time consuming but also error-prone.

High Scalability: Big data applications require high throughput and low-latency data access within DCs. At the same time, huge volume of data will be brought into DC facilities, driving DCs to grow into super-large-scaled platform with sufficient processing capability. It is essential to enable DCs to maintain acceptable performance level when ultra-large-scaling is conducted. Therefore, high scalability should be a critical feature that makes a DC design competitive for the big data era.

Open, Standard based and Flexible Service Layer: Without unified enterprise design for dynamical resource management at different architecture or protocol layers from IO, storage to UI, it makes resources hard to be dynamically allocated based on the time and location sensitive characteristics of the application or tenant workloads. Open and standard based service-oriented architecture (SOA) has been proven effective, to which it enables enterprises of all sizes to design and develop enterprise applications that can be easily integrated and orchestrated to match their ever-growing business and processing improvement needs.

Green/Energy Efficiency: To enable large-scale DC applications in a green and environment friendly approach, energy efficient components, architectures and intelligent power management should be included. The use of new mediums for computing, memory, storage and interconnects with intelligent on-demand power supply based on resource disaggregation help achieving fine-grained energy savings. In addition, intelligent energy management strategies should be included in the DCs, which can track the operational energy costs associated with individual application-related transactions, to figure out key factors, conduct energy-saving scheduling, and tune the energy allocations according to actual demands to dynamically adjust the power state of servers, and etc.

III. OVERVIEW OF ADARM FRAMEWORK

Tremendous amount of research have been conducted that aims at improving the resource utilization. But existing works are focused mainly on "single-point", and almost exclusively at "task level" with CPU-centric and virtual machine oriented scaling technologies. There is a lack of concerted effort with a systematic/holistic approach that can address the application driven, adaptive and integrated fine grained resource management schemes, while considering real networking capability constraints are deemed necessary.

ADARM provides an unified architecture with 2 levels of application-driven global resource allocation optimization infrastructure and task level scaling technology through innovative fine-grained and unified resource management and scheduling optimization techniques. It is aimed to address the application driven and adaptive resource management, especially allocation and scheduling optimization for large scale data centers in near real time.

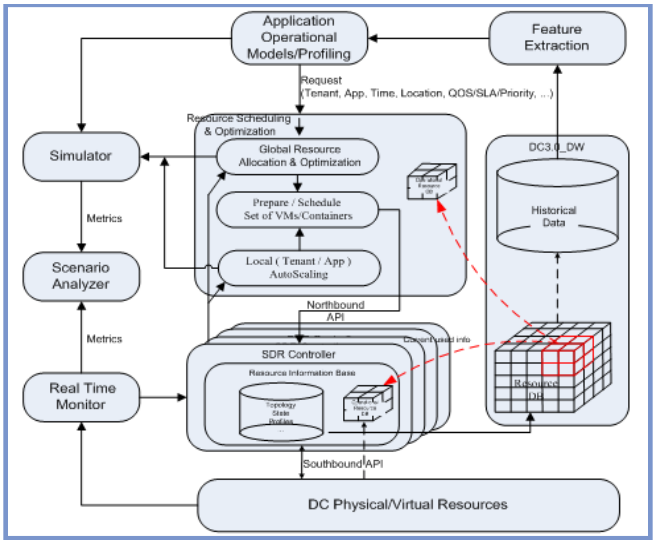

Figure 2: ADARM Framework

The ADARM framework is depicted in Figure 2, and its major components are described below.

A. *Application Modeling and Profiling*

Current DCs pre-allocate resources by aggregated or expected peak demands for every tenant, it can lead to wasted resources over most time and very low resource utilization on average. However, certain applications still could not get enough resources when the upper limit is reached and it could not be easily provisioned in real-time especially when those resources suddenly become scarce as applications compete for resources fiercely. And it is even more so if such resources are not properly allocated in the first place (such as the network bandwidth over some critical links or paths, while VMs or containers are deployed).

As discussed in [2], current application patterns present a complex behavior characterized by time variance, location dependence, and data dependence over an intrinsically distributed network. The proposed ADARM makes on-demand resource decisions based on patterns of the tenants' applications rather than allocating a fixed amount of resources sufficient for some expected peak resource requirement.

The joint application profiles of ADARM, in its simplest form, is just a collected "representative" set of resource requests by every considered application or the application components on when, where, and how much a specific resource need is to be fulfilled in a timely manner, as if otherwise, it would not meet the QoS commitments for the application. Such profiles can be expressed formerly with formal languages, such as the Topology and Orchestration Specification for Cloud Applications (TOSCA) [3] or in a multi-dimensional and fact-dimension based online-analytical model (e.g. Fig. 3 [4]) that could be easily implemented and integrated with a framework and all its supporting components such as ADMPCF[5]and ADARM.

ADARM simply extends the route-based network centric multi-dimensional model to include all types of the resources (CPU/Core, Memory, Storage, Network Bandwidth). As the projected or historical resource usage information is collected into the DC-DW (data-warehouse), application/component level resource usage patterns can be extracted online or offline, utilizing techniques such as Hadoop [6][7], and utilized by the Application Operational Models/Profiling component to formally establish and revise on-demand resource requests.

B. *Resource Data Warehouse, Features, Analytics, and In-Memory Resource Cube*

One of the foundational components in ADARM is ADARM_DW (DC3.0_DW in Figure 2). It is to store historical tenant/application/flow/packet level records, and various system-wise characteristics being extracted. The combined multi-dimensional modeling and analytics framework, together with the effective use of in-memory computing make such a DW both flexible and high-performance.

Following the OLAP (Online Analytical Processing) principles, all the information are modeled in a hierarchical and dimensional structure. We create "resource facts" (route-facts for the network centric case in ADMPCF) with characteristics across those factors/dimensions, and it can further aggregate or drill down for detailed analysis in order to reveal hidden patterns with some of those factors and their combinations that can be leveraged by the routing and resource scheduling optimization algorithms. Unlike relational databases, OLAP does not store individual transaction records in a two-dimensional, row-by-column format. It uses "cubes", a fundamental multidimensional database structure, to store arrays of consolidated information. The data and formulas are stored in an optimized multidimensional structure, while different views of the data and the calculated "measures" can then be created on demand, in addition to the common pre-computed analytics.

The ResourceDB cube sub-component is used to keep a historical perspective on various resource allocation plans

used or available under changing operating conditions. For example, a network specific pathDB [5] can store, for each tenant/application, the resources allocated at different locations at different times, and it is aggregated to reflect the "typical usage patterns" for such an application. A small portion of the resourceDB is also synchronized with the "real-time" resource DB in the central SDR Manager/Controller component, while such real time slices of the ResourceDB are stored in the in-memory multi-dimensional distributed hashing table (MDDHT) [4].

As illustrated in Figure 3, there are two major components in a MDDHT model, namely facts and dimensions. In MDDHT, the route represents the fact and various factors, in which tenants, applications, time, and locality etc. are its dimensions. By associating the causative factors and their values with the fact, we can view the "route" fact from different perspectives, e.g., Tenant T's route and application A's route, or from the combined perspective of T and A.

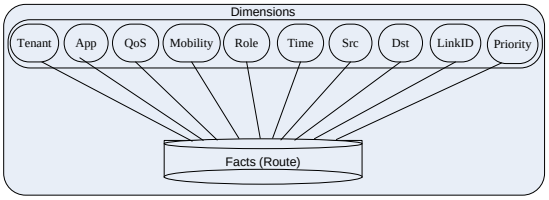

Figure 3: In-memory multi-dimensional Model

Initially during the network planning phase, a substantial set of route entries can be established by the route pre-computation process, as described in [5]. Each path entry is assigned a globally unique identifier by applying a hash algorithm. With MDDHT, normal key/value pairs are extended by adding a measure attribute, named "score", to store the route weights that are defined and preprocessed. Then all the available routes can be sorted according to their associated scores. The (key, value, score) triplet is therefore used to represent the name of dimension, the value of dimension and corresponding route identifier respectively. Thus, route information can be easily associated with a dimension just by adding an entry containing the route identifier to the related dimension tables.

This mechanism can pre-sort and aggregate every dimension based on certain required criteria and rules for certain pre-set dimension values, and automatically reorders with any change of the selected dimension value.

In this way, the MDDHT framework can be very flexible and it enables the use of resource information from different dimensions with multiple perspectives effectively. It can also provide very fast response for retrieving such associated information, to switch among various analytical perspectives dynamically, and analyze them over multiple perspectives comprehensively. For the network centric ADMPCF, once a MDDHT cube is established, ADMPCF can get "good" (or even optimal) forwarding routes satisfying different constraints just by matching various characteristics stored as dimensions in the cube, and for most of the time, it can be done from the stored information without the need to invoke the complex and time-consuming algorithms. In this way, the overall response time will be decreased substantially. With the effective use of available multi-paths, ADMPCF with MDDHT inherently supports network load balancing, and it can achieve higher resource utilization and QoS guarantee with much improved manageability of the expensive network resources, in addition to have better user experience.

C. Resource Allocation & Scheduling Optimization

As depicted in Fig. 2, it is advantageous to profile resource requests from the Application Operational Models/Profiling component across tenants, applications, and even the data center. Such requests contain fine-grained parameters, including Tenant, Application/Component, Time Starting, Ending or Duration, Location, Data Distribution, Network Bandwidth (Origin, Destination), Priority, QoS/SLA metrics etc. They can be stored and updated in the DC_DW. The operational slice can be stored inside the Resource Allocation & Scheduling Optimization (RSO) component and also the (central) Resource Controller.

Two levels of resource allocation and schedule optimization are introduced in RSO: i) application/component level global resource allocation & optimization, and ii) local task level scaling.

1) Application/component level Global Resource Allocation & Optimization

To increase the resource utilization, and therefore, be able to serve more applications and improve QoS with the same infrastructure, resource requests are used to drive the Global Resource Allocation and Optimization (GRAO) component. With the consideration of the joint application profile and with the dependency on location, time, data distribution and QoS requirements, ADARM can significantly outperform the pure VM-based approaches that mostly pre-allocate resources by peak usage demands at the application/tenant level.

By joint profiling and through the global resource allocation optimization, resource requirement variations over time can be utilized to select a proper MRSU dynamically that can increase the overall resource utilization, and/or support the ability to serve additional applications on the same infrastructure. This also shows that our approach can enable autonomous resource allocation and management to significantly reduce the management complexity and cost.

Furthermore, the consideration of joint application profiles, with dependency on location, time, data distribution and QoS requirements, differentiates the proposed approach from the VM-based approach in market place. By joint profiling and global resource allocation optimization, resource requirement variation over time can be utilized to select a proper MRSU that increases the overall resource utilization. This also shows that our approach enables mostly autonomous resource allocation and management, which significantly reduces management complexity and cost.

One instance of such GRAO approach is the Multi-Resource Schedulable Unit (MRSU) [2], where a minimum schedulable unit for a single multi-tenant data center comprises a certain amount of computing, memory, storage, networking resources that jointly could be scheduled to serve a minimum scale application with required QoS.

In general, MRSU coarsely and optimally allocates resources throughout the data center based on estimated/predicted workloads by (tenant, application/component, location, time, data distribution, QoS, etc. The optimization algorithm strives to minimize over-allocation of resources over each time interval (obtained a priori from application profiles), while satisfying all capacity, fairness, prioritization, and resource requirement constraints. An optimal allocation is sought in terms of the specification (or definition) of the minimum schedulable unit (MRSU), and also the allocation matrix that contains the number of such MRSUs per tenant/application based on the following objectives (at least): i) over-allocation should be minimized; ii) minimum/maximum fairness should be guaranteed; and iii) priorities should be respected in the allocation.

With MRSU, the application profiles are combined across applications and across tenants so that a joint profile is utilized to derive the data center-wide MRSU.

At the initialization or proactive planning phase, the Resource Scheduling & Optimization component (RSO) can pre-compute using GRAO, for every planned/scheduled or predicted tenant/application, leading to a set of resource allocation plans tailored to meet resource requirements of different scenarios for each major (origin, destination) pair with multiple critical paths (as for the network bandwidth related dimension) and allocate resources for the projected application traffics, comprehensively utilizing the traffic characteristics, network topology, resources (such as number of CPU, memory units, and network bandwidths), and states. Those resource plans and resource facts, together with their dimensional information, will all be saved into DC-DW and information needed for on-demand allocation can be stored in the MDDHT.

For example [5], in order to make the corresponding network resource scheduling and optimization component ADMPCF adaptive to various changes in the network in real time (or close to real time), some of the algorithms will be run in parallel in order to continuously find better paths, and to assess consequences when certain resources get re-assigned, and trigger some necessary re-optimization based on the global network information. When such situation happens, the results will be refreshed back into the real-time path cube. The key concept behind RSO/ADMPCF is to find good resource plans/routes that satisfy valid constraints from the ResourceDB/PathDB, without the need to invoke the time-consuming optimization algorithms. This is especially critical for managing large systems/networks with thousands or tens of thousands of nodes for large enterprise data centers. In general, when ADMPCF receives a new flow forwarding request, it will first identify all relevant information, then it queries the MDDHT cube for existing "good" route while satisfying all the constraints. Only when it fails to find a proper one, or the overall system performance metric starts to deviate from the projected one by a certain threshold, it would then trigger the re-optimization process.

2) Local Task-Level Resource Scaling

While GRAO tries to allocate resources for ALL applications/components based on the joint profiling and behavior patterns and runs either periodically or when significant enough variations in such patterns dictate a re-optimization of the resource allocation, in which the Local Task/Job level Scaling (LTS) is used to automatically fine tune resource allocations among a set of tasks that can be executed with the same set of MRSUs. Such optimization and planning can allow resource surge from one task to temporarily taken un-used resources from other tasks without adversely impacting overall QoS commitments. LTS is based on an enhanced reinforcement learning auto-scaling [8] algorithm that can learn the behavior of the multi-tier applications/components while automatically adapting to changes. By limiting its applicability to a group of tasks that share a common set of MRSUs, LTS overcomes the well-known state-space explosion problem [8] of such an approach.

D. Service-Oriented APIs for Fine-grained Resources

The northbound APIs of the SDR Controller in Figure 3 is based on RON (Resource-Oriented Network) [9][15] that can abstract and normalize the access to heterogeneous cloud resources at any application defined granularity. Conceptually, RON consists of 3 planes (layers) as shown in Figure 5:

- Data: This bottom plane consists of heterogeneous cloud resources (C-resource), including CPU, memory, storage, network, switches, routers, tasks, jobs, servers, clusters, etc., that communicate in a variety of network protocols.
- Control: This middle plane consists of connected REST resources (R-resource) implemented in any programming languages that expose uniform interfaces and interact with the data plane through their APIs.
- Representation: This top plane consists of hypertexts that represent the states and relations of the REST resources in the control plane with the structures and relationships.

The separation of control and data planes can be logical when the R-resources and C-resources run in the same processes. The control and representation planes constitute the REST APIs of a RON. A RON is designed to have the following properties:

- Access Transparency: The REST resources at the control plane provide uniform interfaces (e.g. HTTP operations GET, PUT, POST and DELETE with hypertext encoded in XML or JSON) to access heterogeneous cloud resources at the data plane.
- Location Transparency: The REST resources at the control plane are identified by URI, which is independent of the physical locations or network addresses (MAC or IP) of the cloud resources at the data plane.
- Connection Transparency: The topology of a RON can change at runtime in an unconstrained way and a client can use hypertext-driven navigation to discover the connections from an entry point to the RON.

• Control Transparency: Communications between the cloud resources at the data plane can be determined without going through the control plane, and the communications between the REST resources at the control plane can be determined without going through the representation plane (i.e. the REST clients).

RON enables us to build RaaS (Resources-as-a-Service) [19] Cloud from a structure of an Elastic Machine (EM), replacing the traditional VM based resource planning in our approach. An EM is essentially a RON assembled on demand by the cloud provider based on user's business requirement, datacenter utilization and workload. To accommodate unpredictable workload, the cloud and REST resources in the RON can be replaced at high frequency over time. Due to the transparency properties of RON, such changes will be hidden from the clients. An EM itself can be treated as a REST resource and it can be composed with others recursively.

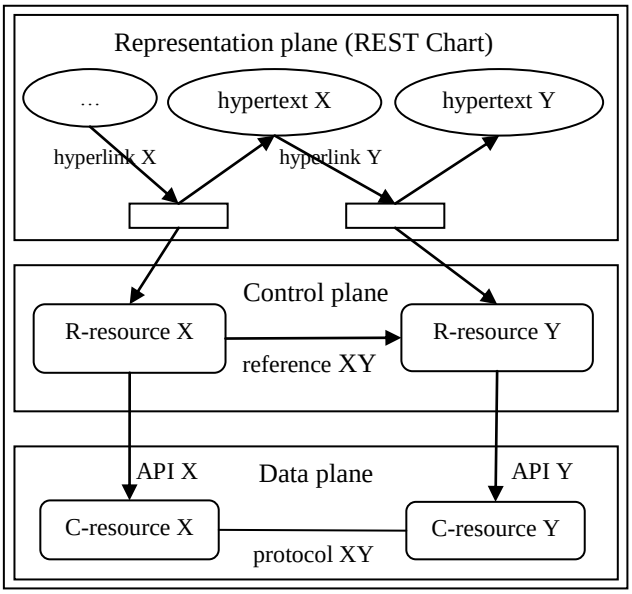

Figure 4: RON Conceptual Model

Based on Docker container and Linux Control Groups (cgroups) technologies, EMs can be described below using a 4 level description:

• Task EM: It consists of a user's Docker container and it exposes a REST API to dynamically control the capacity (CPU, memory, storage and network) of the container.

• Job EM: It consists of a group of related Task EMs which may be collocated or distributed, and it exposes a REST API to control the capacities of the Job EM and the Task EMs.

• Server EM: It consists of aggregated or disaggregated hardware resources, and it exposes a REST API to control the capacity of the Server EM.

The RON REST API protocols are described in details in **Error! Reference source not found.**.

IV. KEY BENEFITS OF ADARM FRAMEWORK

In this section, we will briefly review and discuss some benefits of the ADARM framework for resource management in the future datacenters.

A. Application-Driven and Self-Service

One major reason, that resource utilization has been low for public and enterprise clouds and data centers, is that the resource allocation and scheduling is pre-specified by human administrators or business owners, mostly based on their projection of the peak workloads, and also the inability of the cloud management system to dynamically allocate or reallocate resources as the work load changes.

The ADARM framework addresses these issues by utilizing application modeling/profiling that takes the business and performance requirements of the entire application into consideration when deploying the applications to the cloud. Cloud users can encode their applications in standard mark-up languages, such as OASIS TOSCA [3], and submit the applications to the cloud through self-service portals. The OLAP technologies in our framework can optimally place the distributed components of an application at the datacenter.

B. Software-Defined and Service-Oriented Resources

The ADARM framework implements software-defined data center (SDDC) [11], in which all the infrastructure resources, including hardware and software, are virtualized and delivered as well-defined services, such that the finite cloud resources can be rapidly recombined and repartitioned on-demand using standard remote protocols.

The SDDC goes beyond SDN (Software-Defined Networking) [12] to virtualize fine-grained compute, memory, networking and storage as cloud resources that can be manipulated through software-defined service interfaces. This radical software-defined virtualization results in cost reductions, and it improves the flexibility, elasticity, and resource utilization, in addition to simplify the management operations, troubleshooting, and datacenter control.

C. Adaptive and Optimized Resource Management

Application profiling and modeling in conjunction with the uniform SDDC services enable the ADARM framework to understand the requirements and performance characteristics of the applications, and it allows fast provisioning of the necessary cloud resources for the applications. The application scheduling and optimization modules in the OLAP can use the accumulated application profile and model data to drive the application deployment with horizontal and vertical scaling, as well as application migration in response to workload changes.

Although most resource scheduling and optimization problems are NP-hard, we can use various data-driven machine learning and optimization techniques to find approximate and satisfactory solutions in most cases. The ADARM framework provides an extensible platform for incorporating such algorithms, as the deployed OLAP platform provides the necessary information based on real-

time data and the SDDC service isolates the control APIs from the algorithms.

V. Application Scenarios and Performance Assessment

The major components of the ADARM framework have been prototyped and tested on simulated and actual systems. The following sections describe some of these experiments.

A. *MDDHT Performance[4]*

As networking is probably the weakest link in the DC ecosystem, ADMPCF was developed and experimented first and later extended into ADARM. As discussed before, MDDHT utilizes multi-dimensional online-analytical model with in-memory processing and it significantly improves the controller performance.

First, with a 512 nodes network and the randomly generated traffic matrices, a series of experiments were designed to study how MDDHT would perform. The user interface of the prototype is shown in Figure 5.

Flow requests were pushed from the traffic matrix with a total of 9610 network resource requests one by one or by groups, or manually specify a flow with certain parameters. While those flows were injected into the network, links would get more utilized, and the color coded link utilization would tell the number of links in a certain utilization range from the UI. Underneath that, some key performance indicators, such as average link utilization, percentage and number of rejected requests were updated and displayed continuously. At the bottom, links with top or bottom utilization were also displayed for management review and action.

All the tests were conducted 10 times with randomly generated traffic matrices, and the average performance was reported. The results verified and confirmed the ability and effectiveness of the proposed MDDHT.

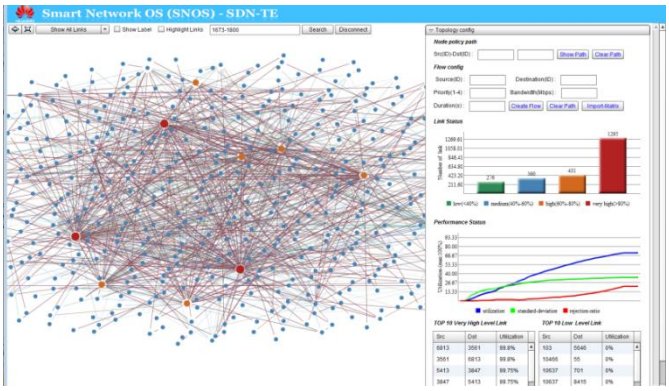

Figure 5: The user interface of the prototype

Figure 6 shows the delay of MDDHT compared to other approaches. We can see that the delay from the traditional relational database implemented with H2 is over 10 times longer than MDDHT. As the number of associated tables increases, such delays would grow exponentially. Normal distributed hashing (DHT) reduces the delay compared with the traditional relational databases, but still is over 6 times longer than MDDHT. As the number of associated column family increases, the delay would also jump.

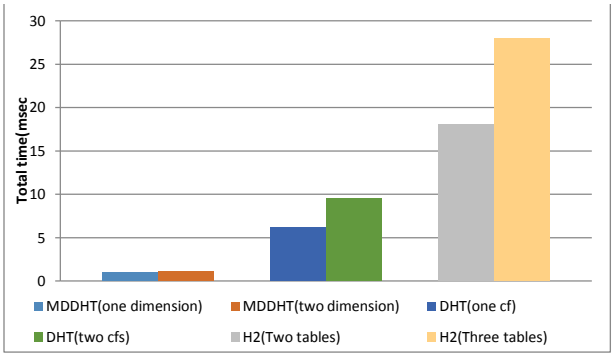

Figure 6: Delay comparison of different approaches

More interestingly, Figure 6 also shows that the number of associated dimensions did not affect the performance for MDDHT.

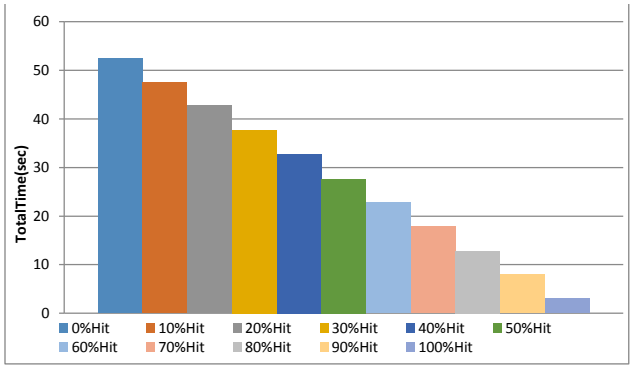

Figure 7: Total time of 9610 traffic matrix requests

Figure 7 illustrates the request processing delay comparison with 9610 new requests in the traffic matrix, while percentage of path hit from the PathDB in ADMPCF varies. As it can be seen, when the percentage of requests needed to call the multi-path algorithm decreased, the delay is reduced almost linearly.

B. *MRSU Performance*

As discussed in [2], an MRSU is a minimum multidimensional schedulable unit for a single multi-tenant data center, consisting of a certain amount of computing, memory, storage, and networking resources that are jointly schedulable and can serve a minimum scale application with required Quality-of-Service (QoS). A minimum combination of all types of resources is selected with the objective that all the tenants' demands can be expressed as multiples of the MRSU. MRSU determines the minimum amount of resources in each dimension that can be configured as a unit, and then the number of such units that should be assigned to an application/component. We have formulated the problem of MRSU allocation as an optimization program trying to minimize over-allocation and a solution inspired by the weighted fair queue algorithm is proposed.

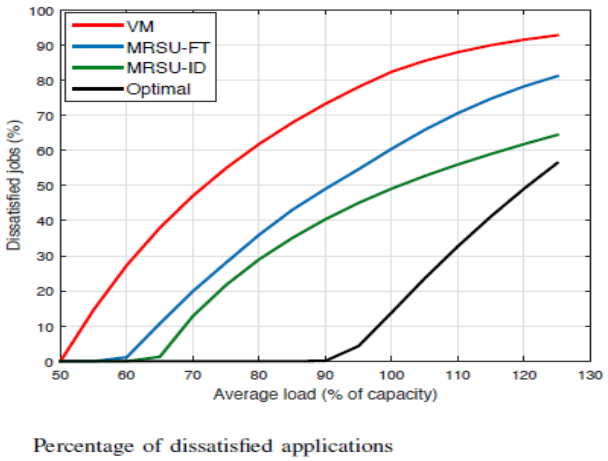

Figure 8: MRSU performance

The MRSU approach is different from the prevailing VM based resource allocation mechanisms in today's DCs, which treat each type of fine-grained resource, such as CPU, memory, storage and network, as independent dimensions.

The performance evaluation results in Figure 8 show a significant performance improvement of MRSU over conventional schedulers based on VMs in terms of client dissatisfaction rate on different workloads. Overall, the current MRSU implementation can handle around 40% more load than VM for the same dissatisfied rate. However, as can be seen in Fig. 8, there is still big gap toward the real optimal allocation. As we continue to improve the algorithms, we would expect still better benefits.

C. SDDC Service Performance

The described EM/RON architecture and RON REST APIs have been implemented in Java based on Docker container on Linux servers. The experimental environment for both prototypes involved a Linux server machine running Docker containers hosting the Task EM/RON within a Tomcat server and a Window 7 client machine running JMeter that emulate concurrent clients.

We used JMeter to simulate 10 concurrent REST clients, while each REST client sent 20 different GET and PUT requests to the RON REST APIs. The average client response time (millisecond) and the total RON based REST API system processing throughput (#request/s) were calculated. In each session, we also recorded the REST API processing time (millisecond) for each request at the server side. To test the task workload impact on the REST API performance, we ran 7, 22, and 32 Docker containers to simulate the task loads on the same Linux server, whereas the RON REST API server was running inside the separate dedicated container.

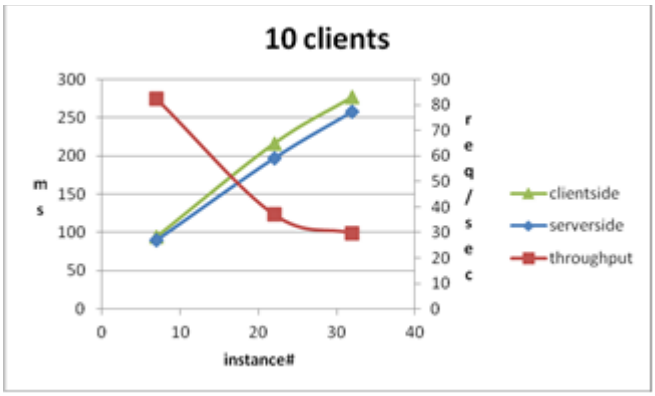

Figure 9: Performance of RON API with 10 concurrent clients

Figure 9 shows performance for 10 concurrent clients. The average client response time increased from 94ms to 216ms and 277ms respectively, for 7, 22, and 32 task containers. The gap between the average client response time and server time is 14ms, which accounts for average network latency and Tomcat processing time. In this experiment, as the number of concurrent client increased from 5 to 10, the overall RON REST API server system throughput increased by 31% on the average, while the average client response time increased by 58% and the average server processing time increased by 53%.

VI. CONCLUSIONS

In this paper, we described the ADARM framework for public/private cloud and large scale data centers (DCs). It provides a unified architecture with 2 levels of application driven global resource allocation optimization and the task level AutoScaling. It is aimed at to address the application driven and adaptive resource management, especially allocation and scheduling optimization for large scale data centers in near real time, with innovative fine-grained and unified resource management and scheduling optimization techniques.

Although the prototypical studies were promising, there is a lack of operational data from large scale data centers, and the applicability and effectiveness of ADARM is yet to be further proven. In addition, some internal mechanisms of the proposed approach could also be improved for better scalability, reliability and performance. Finally, as we move toward distributed control and management over a geo-physically scattered network, the proper balancing act for synchronization, consistency and performance will need to be further studied, especially for large scale distributed data center networks.

REFERENCES

[1] Shannon Lab, Huawei, "High Throughput Computing Data Center Architecture - Thinking of Data Center 3.0," (http://www.huawei.com/ilink/en/download/HW_349607), International Symposium on Computer Architecture (ISCA), Minneappolis, June 14-18, 2014.

[2] David M. Gutierrez-Estevez, Min Luo, "Multi-resource Schedulable Unit for Adaptive Application-driven Unified Resource Management in Data Centers," IEEE International Telecommunication Networks

and Applications Conference (ITNAC) 2015, Sydney, Australia, November 2015.

[3] Derek Palma, Thomas Spatzier (eds), Topology and Orchestration Specification for Cloud Applications Version 1.0, OASIS Standard, 25 November 2015.

[4] Min Luo, Xiaorong Wu , Yulong Zeng, Jianfei Li, Ke Lin, Man Bo, and Wu Chou, "Multi-dimensional In-Memory Distributed Hashing Mechanism for Fast Network Information Processing in SDN," the 9-th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), July 2015.

[5] Min Luo, Yulong Zeng, Jianfei Li, Wu Chou, "An Adaptive Multi-path Computation Framework for Centrally Controlled Networks," Journal of Computer Networks, Elsevier, February 2015.

[6] Long Qian, Bin Wu, Renbo Zhang, Weiyu Zhang, Min Luo, "Characterization of 3G Data-plane Traffic and Application towards Centralized Control and Management for Software Defined Networking", 2013 IEEE Bigdata Congress, Santa Clara, July 2013.

[7] Yuanjun Cai, Bin Wu, Xinwei Zhang, Min Luo, Jinzhao Su, "Flow Identification and Characteristics Mining from Internet Traffic with Hadoop", 2014 International Conference on Computer, Information and Telecommunication Systems (CITS), July 7-9, 2014, Jeju Island, South Korea.

[8] Pradeep Padala, Aashish Parikh, Anne Holler, Madhuri Yechuri, Lei Lu, Xiaoyun Zhu, "Scaling of Cloud Applications Using Machine Learning", VM Technical Journal, Summer 2014

[9] Li Li, Tony Tang, Wu Chou, "A REST Service Framework for Fine-Grained Resource Management in Container-Based Cloud," IEEE Cloud 2015, pages 645-652, New York, New York, USA, June 27-July 2, 2015.

[10] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir: The Rise of RaaS: The Resource-as-a-Service Cloud, Communications of the ACM, Vol. 57 No. 7, Pages 76-84, 2014.

[11] SDxCentral, "What's a Software Defined Data Center?", https://www.sdxcentral.com/sdn/definitions/software-defined-data-center/

[12] Open Networking Foundation, "Software-Defined Networking (SDN) Definition", https://www.opennetworking.org/sdn-resources/sdn-definition

[13] Jiafeng Zhu, Weisheng Xie, Li Li, Min Luo, Wu Chou: Software Service Defined Network: Centralized Network Information Service, IEEE 2013 Software Defined Networks for Future Networks and Services (SDN4FNS), pages 1-7, Trento, Italy, November 11-13, 2013.

[14] Min Luo, "Software Defined Resources for Future Mobile Services and Cloud," Invited talk at 2014 IEEE Conference on Cloud Computing, June 27- July 2, 2014, Anchorage, Alaska.

[15] Li Li, Wu Chou, Min Luo, "A Rest Service Framework For RaaS Clouds," Service Transaction on Cloud Computing (ISSN 2326-7550), Vo. 3, No. 4, October-December 2015.