Computing at Massive Scale: Scalability and Dependability Challenges

Renyu Yang[†] and Jie Xu[*†]
[†]School of Computing, Beihang University, Beijing, China
*School of Computing, University of Leeds, Leeds, UK
yangry@act.buaa.edu.cn; j.xu@leeds.ac.uk

***Abstract* —Large-scale Cloud systems and big data analytics frameworks are now widely used for practical services and applications. However, with the increase of data volume, together with the heterogeneity of workloads and resources, and the dynamic nature of massive user requests, the uncertainties and complexity of resource management and service provisioning increase dramatically, often resulting in poor resource utilization, vulnerable system dependability, and user-perceived performance degradations. In this paper we report our latest understanding of the current and future challenges in this particular area, and discuss both existing and potential solutions to the problems, especially those concerned with system efficiency, scalability and dependability. We first introduce a data-driven analysis methodology for characterizing the resource and workload patterns and tracing performance bottlenecks in a massive-scale distributed computing environment. We then examine and analyze several fundamental challenges and the solutions we are developing to tackle them, including for example incremental but decentralized resource scheduling, incremental messaging communication, rapid system failover, and request handling parallelism. We integrate these solutions with our data analysis methodology in order to establish an engineering approach that facilitates the optimization, tuning and verification of massive-scale distributed systems. We aim to develop and offer innovative methods and mechanisms for future computing platforms that will provide strong support for new big data and IoE (Internet of Everything) applications.**

Keywords — Cloud computing, Computing at Scale, Dependability, Performance, Scalability, Service engineering

I. INTRODUCTION

Cloud datacenters are large-scale distributed computing systems, typically implemented using commodity hardware, capable of provisioning services to various consumers with diverse business requirements. The batch processing and real-time analysis of big data are two of the most exploring and important requirements for Clouds. In the big data era, the volume and velocity of data generation are unprecedented. According to a study by Harvard Business Review [7], over 2.5 exabytes of data are generated every day, and the speed of data generation doubles every 40 months. Through the computing platforms operated by Alibaba in China, hundreds of millions of customers visit the systems every day, looking for things to buy from over one billion items offered by merchants. Hundreds of terabytes of user behavior, transaction, and payment data are logged by the systems, and they must go through daily elaborated processing to provide timely support to the optimization of core business operations (including online marketing, product search, and fraud detection) and decision making, so as to improve user experiences such as personalization and product recommendation. In addition, there are a large number of tracing logs generated within these processes every day. These system logs are extremely valuable both during the development stages and at the operational stage for monitoring, debugging system's operational behavior, and understanding its inherent patterns.

In order to utilize and mine the business data or system logs effectively, data processing has been progressively migrating from traditional database-based store and query approaches to distributed systems which can scale out conveniently. However, such systems must be able to schedule hundreds of thousands of tasks per second, while their applications (e.g. running services and compute jobs) have to be immune to the increasingly unexpected system failures or unforeseen interactions. Therefore, *scalability* and *dependability* have become two fundamental challenges for all distributed computing at massive scale. Despite many recent advances from both academia and industry, these challenges are still far from settled, especially when a system scale grows to over 10K servers and millions of computation tasks.

In particular, the scheduler within such a system can easily become a scalability bottleneck since the system's workloads increase dramatically as the increase of the system's scale. Additionally, a dynamic Cloud system often introduces great heterogeneities due to various user requests and available shared resources. Many specialized systems with different computation purposes co-exist within the Cloud, with diverse resource requirements and patterns. The system resource utilization could be enhanced by running a mix of diverse workloads on the same machines (CPU- and memory-intensive jobs, small and large workloads, and a mix of batch and low-latency jobs), sharing the underlying physical resources.

Furthermore, Cloud service providers are constantly under great pressure to provision uninterrupted reliable services and platforms to consumers while reducing their operational costs due to significant software and hardware failures in such scale systems [43]. A widely used means to achieve such a goal is to use redundant system components to implement user-transparent failover, but its effectiveness must be balanced carefully without incurring heavy overhead when deployed – an important practical consideration for systems at sufficient scale and complexity.

978-1-5090-2253-3/16 $31.00 © 2016 IEEE
DOI 10.1109/SOSE.2016.73

In this paper we will present our understanding of the current challenges in this particular area based on the emerging characteristics of massive-scale distributed computing systems. We will discuss both existing and potential solutions in-depth to the problems especially related to scalability and dependability. Specifically, we will introduce a data-driven analysis methodology that characterizes the important patterns in Cloud data centers including resource, workload, performance and failure models, and traces potential performance bottlenecks in a massive-scale computing environment. Based on the profiling information and knowledge generated from the field data analysis, we will then examine and analyze several fundamental challenges and solutions we are developing to tackle them, including incremental resource scheduling and messaging, decentralized scheduling, request handling parallelism, rapid system failover and state recovery etc. We will discuss the design philosophies, considerations and tradeoffs we usually make when implementing these techniques. Most of them are adopted or applied in-progress into the distributed platforms at the Alibaba Cloud systems. We will also outline a data-driven service engineering approach in which we realize a full closed-circle of engineering processes, including monitoring, analytics, alarming, system optimization, and the eventual verification. Finally, we will describe several further research directions not limited to our particular research areas and demonstrate that the proposed approaches and mechanisms could bring great benefits to an extensive range of computer systems, such as mobile computing, *IoV* (Internet of Vehicles), *IoT* (Internet of Things) etc.

The remaining parts of the paper are structured as follows. Section 2 presents the overview of massive-scale computing characteristics. Section 3 specifies the data-driven analysis methodology while Sections 4 and 5 discuss the scalability and dependability challenges and possible solutions respectively. The systematic application of data-driven service engineering is described in Section 6, and the future directions and research challenges are examined in Section 7.

II. Overview: Computing at Massive Scale

Internet-scale or massive-scale distributed computing for data analysis workloads has been widely discussed in recent years and is becoming increasingly common. High operational and maintenance costs within heterogeneous workloads management and resource allocation lead to the philosophy of sharing data center cluster resources among diverse computation frameworks ranging from batch jobs to long-running services. Scheduling such diverse workloads is inherently complex and difficult, especially as the computing cluster size grows rapidly.

To determine the challenges in massive-scale computing, it is necessary to clearly understand the numerous emerging but inherent characteristics within the wide fields including Cloud computing and Big Data processing. As described in Figure 1, we summarize the emerging characteristics and trends of

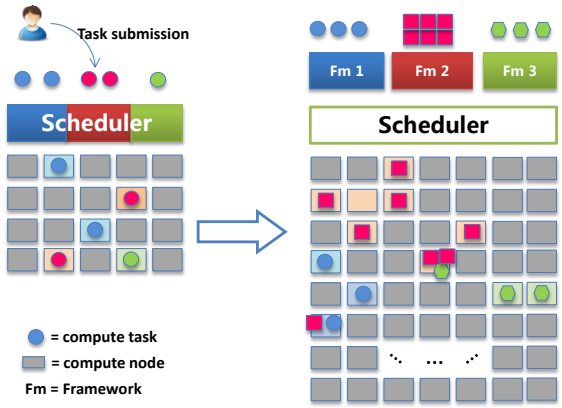

Figure 1. Comparison and evolution: traditional system scale v.s massive system scale

modern massive-scale distributed systems, in comparison with traditional systems in the following aspects: heterogeneity, diverse workloads, increasing scale, and frequent failures etc. To deal with these changes, multiple computing frameworks often have to run on a unified scheduler while handling varying requests. The diverse workloads are usually co-allocated to the shared hardware cluster in order to improve utilization.

A. *Varying Request and Resource Heterogeneity*

There are great heterogeneities produced by different user requests and available resources in a typical data center cluster.

Firstly, the request heterogeneity can attribute to the highly dynamic Cloud environment, where users with different computation purposes co-exist with diverse resource requirements and patterns. User-specific attributes can be normally expressed by the required type and amount of resources and other attributes that could dictate detailed preferences, including data locality, the optimal location where a specific workload can be executed, security requirements [26][27], geographical location, or specific hardware constraints such as processor architecture, number of cores or Ethernet speed among others described in [12][13]. Secondly, for the resource heterogeneity, the computing machines that constitute the data center cluster are typically constructed from commodity hardware owing to significantly reduced merchandising and operational costs in comparison with a high-performance supercomputer. The configuration diversities among different machines lead to the huge discrepancies in a cluster. These diversities can be characterized using dimensions such as micro-architecture, machine chipset version, CPU and memory capacities etc. [44]. In fact, the machines are constantly supplemented into the computing cluster over time, using whatever configuration was most cost-effective [13]. Despite these benefits, these commodity servers are very vulnerable to hardware and software failures [43]. Therefore, the Cloud datacenter providers have to be constantly under great pressure to face increasing failures in such systems in order to provision

TABLE 1 STATISICAL DATA DURING 2015 ALIBABA DOUBLE-ELEVEN SHOPPING FESTIVAL [2]

Type	Number
Peak order number	Over 120,000 per second
Total payment transactions in *Alipay*	710 millions
Peak payment transactions in *Alipay*	85,900 per second
Peak transactions processed on *AliCloud (Alibaba Cloud)* platform	140,000 per second

TABLE 2 STATISICAL DATA OF ONE PRODUCTION SYSTEM IN ALICLOUD.

Type	Number
Server number	4830
Job number	91,990
Task number	42,266,899
Worker number	16,295,167

uninterrupted reliable services to their consumers. The mentioned heterogeneities will increase the scheduling complexity since the system has to pre-filter the candidate targeted servers for the specific request in the waiting queue by going through whole search space of available machines according to the request constraints and specification.

B. Workload Diversity and Resource Sharing

With the prevalence of big data concepts and techniques, the demands for data analysis and processing increase dramatically. At present, cluster computing systems are increasingly specialized for particular application domains and purposes. Generally, we can categorize the workloads into online services and offline processing. Online services can be regarded as long-running services, such as virtual machine rental, email service, storage service etc. For the offline processing, in addition to early systems such as Map Reduce [18], Dryad [17], more and more specialized systems for new application domains are emerging both in academia and industry. In particular, these systems include: Spark for in-memory computing [34], Storm [31] and MillWheel [45] for stream processing, Dremel [46] and Hive [47] for interactive SQL queries, Pregel [48] for graph processing, Tez [50] and FuxiJob [28] for DAG processing, and GraphLab [49] for machine learning etc. Although these systems seem to be a natural way to achieve their corresponding computations effectively, these solutions can achieve neither *high server utilization* nor *efficient data sharing* [33]. In reality, most cloud facilities operate at very low utilization [38]. It seems contradictory to the fact that some clusters might be very busy or be extremely short for a specified resource dimension (such as CPU-, memory-intensive), although other separate clusters are idle but cannot be fully utilized by others. The data sharing among different frameworks on separate clusters becomes difficult and have to leverage data exporting, replicated to permanent storages for temporary buffering.

Consequently, high operational costs force heterogeneous applications to share cluster resources for achieving economy of scale. The highly-required utilization and data sharing demands motivate the system evolution to support multi-tenant workloads in a unified system in order to improve the efficiency and utilization.

C. Increasing Request and Cluster Scale

According to a bankcard analysis [6], Visa Card system processed over 24,000 transactions per second in 2010 while more than 160 million transactions per hour could be done inside Master Card system in 2012. In comparison, the upper payment transactions in *Alipay* (*Alibaba Group*'s payment system) even reached 85,900 transactions per second during the double-eleven shopping festival in 2015 [1][2], which had surpassed visa as the most transacted payment gateway. Table 1 illustrates the peak or total throughput of systems in Alibaba [2]. The throughput is indicated by the transactions processed in the payment subsystem or big-data processing system. All order and transaction data are finally extracted into a large-scale computing system for real-time processing and business analysis. Specifically, a transparent user experience is highly desirable during the request bursting period without noticeable response latency or service timing-out due to the overloaded workloads beyond the system capacity. Therefore, the high-stress requests and transactions require the underlying systems to cope with them timely and keep the waiting queue size as short as possible. Subsequently, effective resource assignments and allocations are expected in order to accelerate the turnovers of system resources, thereby improving the resource utilization.

Meanwhile, the increasingly enlarged cluster size also gives rise to difficulties of cluster management and the increasing scheduling complexity. At present, Yahoo reported that they can support up to 4,000 nodes before YARN [37]. Alibaba had supported 5,000 nodes resource management and provision [28] and will support over 10,000 nodes recently. Google claimed that their Borg system can run workloads across tens of thousands of machines effectively [36]. In reality, with the cluster size increased, even the periodical status updates and reports carried by the heartbeats would become a heavy burden, leading to the message congestions. The RPC call might be invalid when messages are aggregated within the sending queue, resulting in severe package drop and loss. The messages re-sending retries will further aggravate the system handling capability and the system might eventually hang out and fail to handle any request.

Consider a cluster with hundreds of thousands of concurrent tasks, running for tens of seconds on average, the resource demand/supply situation would change tens of thousands times per second. Making prompt scheduling decisions at such a fast rate means that the resource allocation must realize a rapid and relatively precise *mapping* of the CPU, memory and other resource on all machines to all tasks within every decision making. The statistics shown in Table 2 give a brief example of the numerical data, including the computational tasks and the available compute nodes in a typical production system at massive scale [28].

D. Frequent Failure Occurrence

Massive-scale systems are typically composed by hundreds of thousands to millions of alive and interacting components comprised by the resource manager, service framework and

computational applications. With increasing scale of a cluster, the probability of hardware failures also arises [20][21]. Additionally, rare-case software bugs or hardware deficits that never show up in a small-scale or testing environment could also suddenly surface in massive-scale production systems. Essentially, failures have become the norm rather than the exception at large scale [19]. Due to such system scale, heterogeneity and complexity, it is very likely that different types of faults will manifest. According to our observations, there are a variety of failures causes including halt failures due to OS crash, network disconnection, and disk hang or insufficient memory (OOM) due to bugs in codes, overweight system utilization, performance interference, network congestion etc [22][28][64][65]. As we discussed early, the servers adopted widely in Cloud datacenter use commodity hardware, resulting in deteriorating situations. At the same time, the increased cluster size itself introduces much more uncertainties and reduces the overall system reliability, largely due to the increased failure probability of each node and software component. In this environment, traditional mechanisms such as health monitoring tools or heartbeat tracking can help but cannot completely shield the failures from running applications. Fault tolerance is an effective means in enhancing the dependability of Cloud systems, and will ultimately reduce the economic impact and service degradation for providers and consumers respectively.

Based on fundamental analysis above, we will mainly focus on two outstanding system problems which are urgently needed to be addressed – scalability and dependability in this paper.

III. DATA-DRIVEN METHODOLOGY

Establishing a good understanding of Cloud systems is of increasing importance as well as complexity due to a Cloud's ability to elastically scale-up and down provisioned resources on-demand [10]. Additionally, such systems need to satisfy the expected Quality of Service (QoS) requirements to fulfill the diverse business objectives demanded by consumers. As a consequence, it is a crucial requirement to characterize the workloads running within a Cloud environment.

Analysis and simulation of Cloud tasks and users significantly benefit both providers and researchers, as it enables a practical way to improve data center functionality and performance. For providers and system developers, it enables a method to enhance resource management mechanisms that effectively leverage the diversity of users and tasks to increase the productivity and QoS of their systems. For example, we exploit task heterogeneity to minimize performance interference of physical servers or analyze the correlation of failures to reduce resource consumption. It is also extremely useful for us to find the potential system deficiencies and bugs according to the daily regression testing and profiling data analysis.

In our previous works [14][15], we conducted the comprehensive analysis at cluster and intra-cluster level to quantify the diversity of Cloud workloads and derive a *workload model* from a large-scale production Cloud data center [8]. The presented analysis and model capture the characteristics and behavioral patterns of user and task variability across the entire system as well as different observational periods. We further quantify the *interference-energy model* in which we comprehensively analyze the energy-efficiency of massive system impacted by performance interference [41] and *failure-energy model* which depicts the energy-efficiency reduction and wastes due to constant failures in the Cloud data center [43].

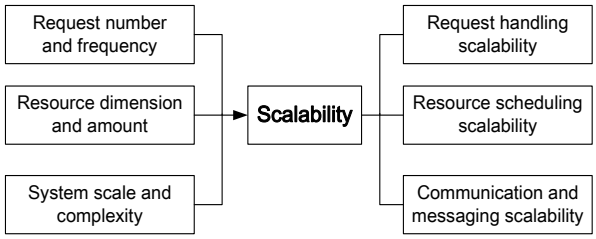

Figure 2. Charateristics of massive-scale system scalability, challenges and the main concerns.

The data-driven analysis is critical to improve resource utilization, reduce energy waste and in general terms support the design of accurate forecast mechanisms under dynamic conditions with QoS offered to customers improved. For example, we classify the incoming tasks based on their resource usage patterns, pre-select the hosting servers based on resources constraints, and make the final allocation decision based on the current servers performance interference level [40][41]. Additionally, we propose a practical data engineering method which uses the data analytics methodology to driven an automatic service for system monitoring and diagnosis, thereby instructing where and how to optimize and improve the system.

IV. SCALABILITY

A. Challenges

Resource scheduling can be simply considered as the process of matching demand (requests to allocate resources to run processes of a specific task or application) with supply (available resources of cluster nodes). Therefore the complexity of resource management is directly affected by the number of concurrent tasks and the number of server nodes in a cluster. Additionally, other factors also contribute to the complexity, including supporting resource allocation over multiple dimensions (such as CPU, memory, and local storage), fairness and quota constraints across competing applications; and scheduling tasks close to data.

To deal with the increasing explosion of running tasks and the cluster scale, computing systems at massive scale have to firstly take the scalability issues into considerations. In this context, we define *scalability* as a constant system capability that sustains the scheduling throughput (such as the operation per second) whilst controlling the perceptional response

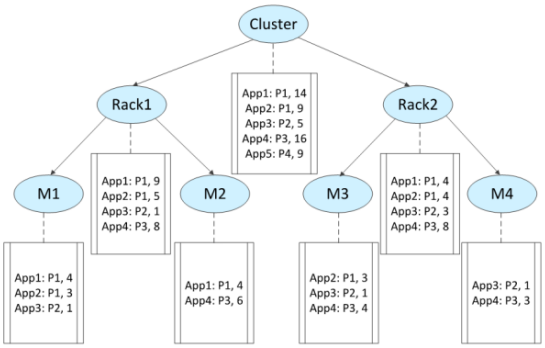

Figure 3. Locality-tree based incremental scheduling example. For example, App1 totally requires 14 units of resources in the cluster, and prefers 4 units on M1 and 4 units on M2 with the highest priority P1 due to data-locality considerations.

latencies as if in ordinary smaller scale. We describe the characteristics and challenges in Figure 2.

To understand the mechanisms of scalability, we divide the issues into the following aspects:

[S1:] Request handling scalability - Running workloads (such as job or application) will propose resource requests to ask for resources according to application-specific execution logic. The requests will aggregate if they are not handled timely by the general resource manager. To avoid the request aggregation, the system should provide high cluster throughput with low-latency request handling and allocation decisions.

[S2:] Resource scheduling scalability - Making prompt scheduling decisions at such a fast rate means that the resource allocation must realize a *mapping* of the CPU, memory and other desirable machine resource to all tasks within every decision making.

[S3:] Communication and message scalability - In general, internal scheduling related instructions or states exchanges in most massive-scale systems are suitably piggy-backed by periodical heartbeats or interactive messages. A long period could reduce communication overhead but would also reduce the utilization when applications wait for resource assignment. On the other hand, frequent adjustments would accelerate the response to demand/supply changes, resulting in promotions of system resource turnovers and throughputs; however, it will cause the *message flooding* phenomenon. Thus, how to properly control the messaging amount whilst maintaining the scheduling performance is a great challenge.

B. Solutions

(1) Architectural Evolution

The architecture experienced several phases:

a) Single-master phase - A naive approach is to delegate every scheduling decision, state monitoring and updating all in a single master node (such as the JobTracker in Hadoop 1.0). But it will be severely limited by the capability of the master and usually leads to *single-point failure*, eventually negatively affecting the system dependability.

b) Two-level phase - this type of approach decouples the resource management and the framework- or application-specified scheduling into two separate layers. For example, Mesos [33] adopts *offer-based* philosophy, provisioning a calculated resource to each upper framework according to dominant resource fairness. In comparison, Yarn [37] and Fuxi [28] utilize *request-based* approach, in which the central resource manager is responsible for resource negotiation among different resource requests and application master takes charge of job scheduling. It significantly mitigates the loads and stress on the central master while enables a customized and flexible resource requirement in the meantime.

c) Decentralized-schedulers phase – the third evolution to improve the scalability is decentralization. In general, multiple distributed scheduler replicas are adopted via multi-threads or independent processes, and each scheduler can handle requests simultaneously based on its local cached states or global shared states [35]. Such typical systems include: Apollo [29], Mercury [30], and Borg [36]. Moreover, no central state need to be maintained if the scheduler (such as Sparrow [32]) adopts batch-sampling and only sends resource probe to find candidate server. This category is particularly effective for those scenarios with a strong low-latency requirement.

(2) Effective Scheduling Approach

Apart from changes derived from system architecture, some scheduling techniques and mechanisms are proposed which can be demonstrated to be very effective and efficient.

Incremental scheduling – Achieving rapid response and prompt scheduling decisions at such a fast rate means that the central resource manager cannot recalculate the complete mapping of CPU, memory and other resource on all machines to all applications tasks in every decision making. In our previous work, we proposed a locality tree based incremental scheduling [28] in massive scale computing and only the changed part will be calculated.

For example, when *{2cores CPU, 10GB Mem}* of resource frees up on machine A, we only need to determine which application in machine A's waiting queue should get this resource. There is no need to consider other machines or other applications. The locality tree will be gradually formed when some of resource requests cannot be handled instantly and have to wait for scheduling. Each resource request will be enqueued into different queues according to its locality preferences. Figure 3 shows a concrete example of the scheduling method. Micro-seconds level scheduling can be achieved in light of this intuitive but effective locality-based approach.

Decentralized scheduling – Decentralized method is mainly aimed to significantly reduce the scheduling latency. We can further classify it according to how states are used:

a) Local state replica coordinated by central master: The functionality of the central master can be simplified to only synchronization all states as a coordinator once the resource or state information is updated by any scheduler [29] [30] [36]. Typically, the used states are derived from load information

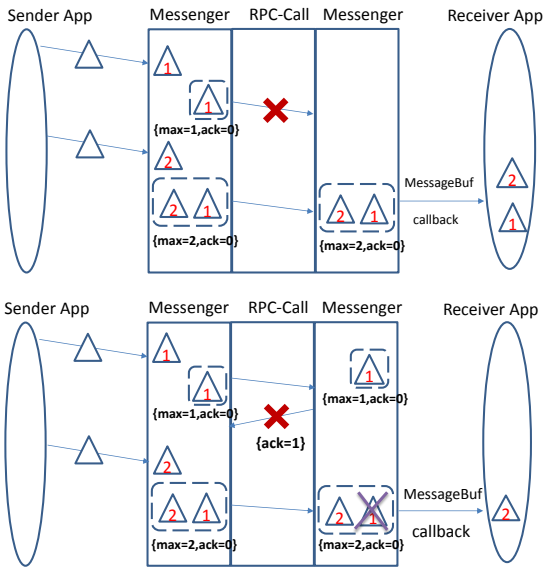

Figure 4. Message re-sending and de-duplication in incremental communication messenger

and abstracted states or metrics, rather than fully cluster and workloads states which are widely-adopted in centralized schedulers. Since each running job performs independent scheduling choices and the task is actually queued directly at worker nodes, the core philosophy is to disperse the burden and potential bottleneck of the central resource manager onto many execution nodes. However, distributed schedulers make local scheduling decisions which are often not globally optimal. Moreover, the state synchronization and conflict resolving must be handled effectively to guarantee that a particular resource is only made available to one scheduler at a time.

b) Shared states visible to all schedulers without a central coordinator*:* Shared states can enable each distributed scheduler full access to the entire cluster and allow them to compete in a free-for-all manner [35]. The communal states can be locked using exclusive locking techniques or lock-free optimistic concurrency control by using *incremental transaction.* To our understanding, inside the transaction an atomic action will be consecutively conducted: the resource assignment decision and the global shared-state updates. The action is in fact equivalent to the states re-syncs with conflicts resolved mentioned in approach a).

c) Stateless distributed scheduling techniques: Another fully-decentralized approach within the spectrum is sampling-based probing for low-latency (e.g., Sparrow [32]). Such designs are highly scalable since there is no requirement to maintain central states and global resource view. There are multiple independent schedulers each of which is responsible for scheduling one or a subset of jobs. Each autonomous scheduler detects servers with fewer queued tasks by probing *m* random servers and assigns the tasks of its jobs to targeted machines in the cluster.

Generally speaking, fully-decentralized solution is indeed very efficient for those latency-sensitive scenarios such as interactive queries. However, this design will be extremely hard to strictly satisfy the scheduling constraints (such as fairness, capacity, and quota management) when only depending on fast-changing global states without high synchronization cost. Therefore, the system designers must strike the balances between the scalability and other variables according to their main objectives.

(3) Effective Message Communications

Incremental Communication – Because of the message flood in the massive scale system, a simple iterative process that keeps asking for unfulfilled resources will take too much bandwidth and get worse when cluster is busy. For this reason, we try to reduce the message amount by only sending messages from running job masters and execution daemons to the central resource manager when changes occur. Only the delta portion will be transferred. Jobs or Applications can publish their resource demands in incremental fashion when the requirement adjusts according to runtime workloads. Consequently, we propose an incremental communication and messenger mechanism. In particular, it should fulfill:

a) Message order-preserving - we must ensure the changed portions be delivered and processed in the same order at the receiver side as they are generated on sender side;

b) Message idempotent resending - we must achieve the idempotency of handling delta messages, which might happen as a result of temporary communication failure;

c) Message deduplication – we de-duplicate the message to minimize the network traffic and avoid useless communications.

An example is demonstrated in Figure 4 and message resending and deduplication will occur when the network package get lost between the sender and the receiver.

Cluster Partition – For the performance and communication scalability, we use multiple replicas of request manager to handle communication and periodical status reports in parallel. A compute cluster can be divided into several *area partition* (the equivalence notion of *link shard* in [36]) and each manager replica is responsible for request handling and information delegation of severs within its specified partition. The consistency will be guaranteed by an elected central coordinator and only the coordinator can conduct changes to the permanent store. Each manager replica will aggregate and compress this information by reporting only differences to the coordinator, in an incremental way as we discussed above.

V. DEPENDABILITY

A. Challenges

Dependability is a key concern for resource managers due to increasingly common failures which are now the norm rather than the exception caused by the enlarged system scale and complexity, different workload characteristics, and plethora of faults types that can activate. Such failures within a massive-scale system have the potential to cause significant

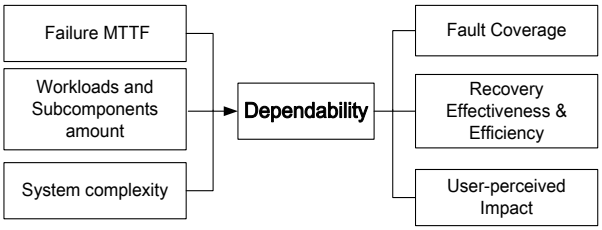

Figure 5. Charateristics of Massive-scale system dependability, challenges, and required specific considerations.

economic consequences to Cloud providers due to loss of service to consumers [9][55], and affect services provisioned to millions globally in the event of catastrophic failures.

Traditional techniques face a number of challenges and will no longer directly suitable to the massive scale systems due to the unaffordable costs and overheads. Specifically, redundancy-based methods such as Recovery Blocks [23], *N*-Version Programming (NVP) [24], *N*-self Checking Programming (NSCP) [25] rely on replicated redundant components, but it is infeasible to apply these redundancies into each component within a system composed of millions of components and jobs. Another widely-used fault-tolerant technique in distributed system is checkpointing. The system can recover its states by restoring the recently-recorded checkpoint logs or files. Checkpoint and Restart (CR) is utilized in high performance computing (HPC) and super-computing (SC) areas, due to the significant reduction of re-computations. In particular, periodical multi-levels checkpointing and rolling-back techniques [54] are suitable for long-running MPI tasks but cannot be properly applied in short tasks or time-sensitive tasks. This is because the resource requests and allocations of these MPI tasks are determined in advance and will not change during its life-cycle. The task number is also not large compared with available resources, making sufficient resources to conduct redundant checkpointing. Another extended application of checkpointing is the snapshot and restore techniques in virtual execution environment [51][52][52]. The availability and dependability of virtual machine and the overall virtual cluster can be guaranteed by recovering the network whilst restoring the memory and disk states from the snapshot file. However, it is relatively time-consuming considering the large amount of runtime memory page size and disk states. In our proposed massive-scale computing system which consists of hundreds of thousands running tasks and active system components, it is extremely ineffective due to the non-negligible additional overheads incurred by conducting checkpoint, taking the disk space, communications and operations into account.

In general, above systems achieve effective resource scheduling and management by large backlogs of pending work - an assumption which cannot be adhere to the on-demand access required for Cloud computing. Considering the large cost for millions of running tasks, it is infeasible to conduct them in Internet-scale systems. Within the context of Cloud resource managers, such techniques are required to effectively scale to thousands of servers, with acceptable overhead and impact to system performance. Thus we summarize the dependability challenges shown in Figure 5 and as follows:

[D1:] Faults and handling coverage - Components within the resource manager are likely to experience different types of faults ranging from crash-stop to late timing failure, as well as have different underlying root causes [65]. As multiple components tend to fail simultaneously and also exhibit correlation, these failures will also complicate the system fault-tolerant solutions. Therefore, we have to maximize the fault coverage from both faults mode and fault handling coverage respectively.

[D2:] Recovery effectiveness and efficiency – The recovery effectiveness can be evaluated by whether the infected component or application can continue to work. In specific data processing context, computation job might fail due to partial subtasks are evicted and re-compute during the recovery. In addition, the recovery efficiency is also a significantly important metric, which might include the full recovery time, the system utilization and the additional resource cost produced by the recovery, the latent negative impacts onto other components or workloads, and the propagation pattern and behavior among different subsystems etc. In a massive-scale environment, all these above will become increasingly complicated due to the shortened MTTF (means frequent failure occurrence), a plenty of component combinations, and system architectural complexity.

[D3:] User-perceived impact – From our experience in massive-scale systems, resource overhead due to eviction, and re-computation of non-faulty workers produces a substantial amount of waste [16]. More importantly, long-running services are disproportionally affected due to restarting worker execution, leading to severely-suffered QoS. Such behavior will also result in increased strain on the resource manager, which has to handle more requests and reschedule workers onto nodes, causing reduced component performance as well as further increased failure probability. Therefore, how to implement a *user-transparent failover* technique to recover the service without noticeable changes to provisioned service perceived by consumers is a big challenge.

[D4:] Easily-used failure detection and diagnosis - In spite of the proposed system prevention or recovery measures, some failures will always occur. The right tools can quickly find the root cause, minimizing the duration of the failure. In addition to the software aging or system failure, human factor errors are observed to be another important provenance [58]. Although our approach can be self-healing in face of non-human causes of errors, manual measures and technical staffs have to get involved if necessary. Therefore, rapid and effective detection and diagnosis approach can ensure a fast access to the types of abnormal metrics.

B. Solutions

To maximize service reliability whilst minimizing detrimental effects to service performance, we propose several fault-tolerant techniques to illustrate a feasible design and

implementation towards reliable service execution for effective computing systems at scale.

(1) Rapid and Effective Component Failover

Failover with reduced checkpointing – we present the philosophy and architecture of a novel approach for component failure recovery that collects and exploits states collected from neighboring components instead of solely relying on hard-state periodically collected from dedicated backup systems. In particular, minimized hard state such as meta-data and information are persistently stored within a node locally, distributed file system or distributed coordination service. Actually, we leverage the distributed memory to store each component states which can constitute the overall system states and be used to recover infected components.

Minimized worker eviction – we achieve it through loose-coupling master or agent behavior from its respective workers during the execution. Specifically, this entails that failure occurrence of a master or agent does not result in its non-faulty workers to be automatically evicted. For example, to tolerate timing failures, the central resource manager attempts to preserve the assigned resource for running workers as if timing-out daemons are still executing rather than directly evicting and re-scheduling them. In this manner, such faults will have minimal interference with perceived reliability.

(2) Optimized Recovery Time v.s. Degraded Service Level

According to our reduced hard state recovery strategy, the additional overhead cost is mainly dependent on the collection and required boundary of state information completeness. Incomplete information might appear due to timing-out components unable to contribute their states in time. The collection time also closely depends on cluster scale, application number, and application-specified configurations. For instance, increased application number signifies a larger amount of states to collect and the requisite time correspondingly. On one hand, longer waiting time can potentially lead to the mitigation of soft states incompletion, but resulting in extra end-to-end recovery time. On the other hand, insufficient collection time leads to incomplete states and subsequent degraded service level (e.g., job extended running time due to worker eviction, or system slow response due to state absences). Thus, it is necessary for cluster administrators to strike the balance between the recovery cost and various levels of degraded service.

(3) Blacklist and alarm dashboard to help diagnose failures

Multi-level blacklist - It is high probable that within the datacenter's lifetime, physical nodes will experience crash or timing failures. Such behavior can result in cascading failures, as well as long-tail phenomenon of application execution [56]. In order to mitigate such occurrences, a multi-level machine blacklist has been designed and deployed in order to detect and isolate faulty nodes from the rest of the system. This blacklist functions by monitoring system behavior at both cluster and application level. The blacklist can be added through system autonomous program or by technical staffs manually according to their experiences and engineering requirements.

More specifically, for the cluster level, a heartbeat is sent between the node daemon and resource manager, reporting the health situation of each node within the cluster. If the manager detects a heartbeat timeout, the node will be removed from the scheduling resource list, and a resource revocation is sent to the application master so that it can evacuate the running instances away from the unresponsive executive nodes. Application-level blacklisting calculates the health of a physical node based on the status of workers as well as failure information collected by the node daemon, and it operates both at task-level and job-level. If one worker of an application has been reported as failed within a node, the node will be placed into the blacklist for the particular task which is currently executing in the worker. This action is taken under the assumption that the faulty behavior of the task could potentially be the result of the task operational requirements to execute on that particular hardware specification.

System health self-checker and dashboard - the operational characteristics of each physical node and internal system components are monitored periodically using a *health checker* tool to diagnose the node health and process status, such as disk statistics machine load and network I/O etc. in order to calculate a health score. If the score falls beneath a specific threshold, the component will mark the node as unavailable. An advantage of this approach is that datacenter administrators are capable of adding customizable check items to the list for specific error detection, and an alarm will be triggered on the monitoring dashboard. The technical staff can be involved and leverage the alarming information to timely find workaround solutions.

VI. CASE-STUDY: DATA AND SERVICE ENGINEERING

For datacenter management, existing methods are tedious, error-prone, and ultimately time consuming [57]; requiring the expertise of a large number highly trained datacenter engineers to develop in-house development scripts, or in the worst case scenario, perform the process of system monitoring, processing, and analysis manually. Therefore, we leverage the data-driven methodology and integrate the depicted massive computing entities model in Section 3 into an autonomous and automatic profiling system to aid decision making. Such decisions include the detection of the system abnormal behaviors, driving the further system optimizations, evaluation of the consequent effectiveness, and finally making configurations refines to the resource management mechanisms deployed within the infrastructure. Figure 6 describes the whole architecture of the proposed closed-loop workflow and it is composed of several core components:

Tracelog collector - We add probes in order to monitor and collect log data of system components. For example, in order to comprehensively monitor the lifecycle of an application, we monitor event status changes (i.e. *submitted, scheduled, running, failed, completed*) and resource utilization of physical nodes and applications. Furthermore, it is also

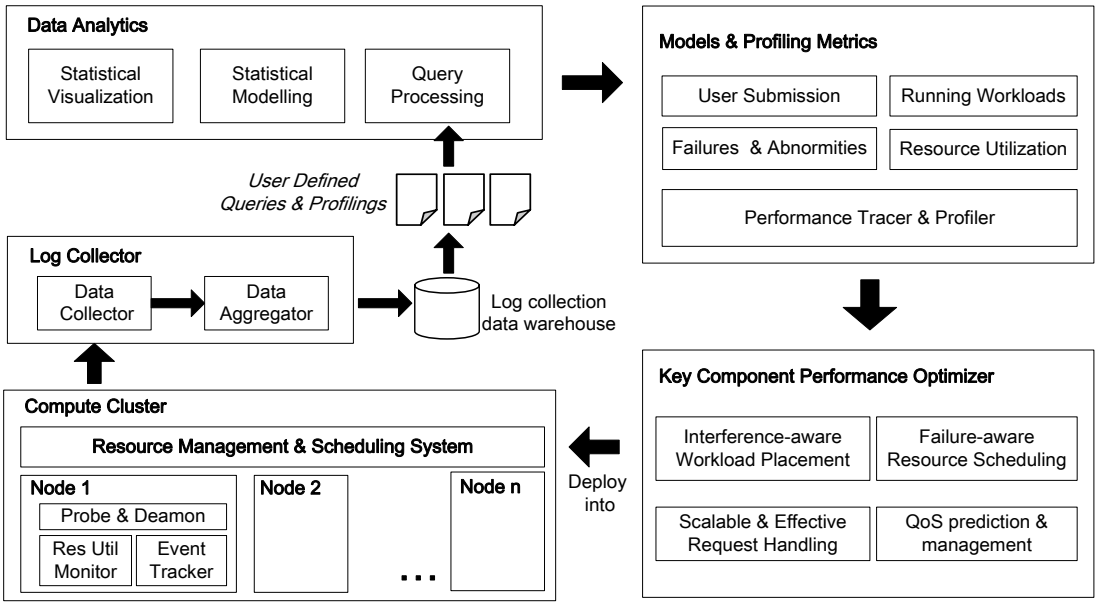

Figure 6. Data-driven methodology and overall closed-cycle of performance monitor, optimization and deployment.

necessary to profile some system metrics and overheads incurred by communication between components, and latency between resource request and negotiation for applications. As a result, how to collect and monitor the generated tracelog efficiently while mitigating its impact of service performance is a big challenge. Our approach uses the *inotify* [59] mechanism in Linux 2.6 in order to incrementally tracelog when there are changes within individual files and directories.

Data analysis engine - In order to exploit the monitored profiling data, we implement an analysis and visualization service based on the Alibaba *Open Data Processing Service (ODPS)* [3]. ODPS is the proprietary data platform in Alibaba providing massive data storage and query processing service. The query processor allows users to extract the results of interest from the collected log data of the cluster. The processor provisions an SQL type language to users which is automatically translated into a DAG workflow for query processing. The generated profiling data could be populated automatically into the data warehouse and customized queries are executed within the ODPS control-plane, by importing the data flow from the data-plane. Additionally, we calculate and conduct statistics-oriented computing based on the outputted results using *R-statistic* programming environment [60], which is an integrated suite of software facilities for data manipulation, calculation and graphical display. In this way, we integrate visualization and mathematical modeling into our service in order to produce charts, distribution modeling and cluster analysis etc.

Diagnosis, tuning and optimization – Based on the data analysis framework, statistical analysis and visualization of the metrics profiler will facilitate the exploration of operational behaviors. Consequently, diagnosis, correction, and tuning to the system configurations or implementations could be conducted. Correction entails a reactive approach of direct intervention by technical staff to perform fault-correction upon the performance metric alarm detection. It allows for technical staff to identify and manually correct potential problems within the system for reducing QoS violations and catastrophic failure prevention (such as system outages). After the optimization, our profiling system can provide an automatic verification and test environment to evaluate the latest updates or configuration changes. The proposed closed-cycle can protects against rapid development and deployment of bad configurations and provides a system-test framework to guarantee the code quality from engineering aspects. In practical, we have used this methodology in our previous works to realize system utilization improvement [42] and request latency reduction [39].

VII. FUTHER RESEARCH DIRECTIONS

Big Data as a Service (BDaaS) - With the blooming of all sorts of big data provenances over the Internet, the huge data volume has become too large and time-consuming for individuals to calculate on personal machine or small-scale servers. The business model in Cloud computing is to enable on-demand and flexible resource provision. Similarly, the big data storage, analytics and management could be integrated together and provided as a service to customers [11]. Typically, customers only need to write their own processing logics according to the BDaaS APIs without any concern in terms of the underlying running location and implementations. In this

context, the scalability and dependability of BDaaS platform are significantly important to guarantee the customer's SLA.

Debugging large-scale distributed applications - The management difficulties of large-scale distributed systems mainly derive from the intricate relationships among different processes (all kind of masters, slaves, and execution workers) that are widely dispersed on different compute nodes, and the extremely large size of the system logs. Debugging or investigating a distributed application performance issue or system bugs usually needs to search for some specific key events information from the massive logs. Due to the semi-structured or unstructured log information, the heterogeneity will lead to an inability to produce a single unified query or scheme for issue diagnosis. From our industrial engineering experiences, it is extremely time-consuming for engineers and technical staffs to find root-causes of problems in the production clusters or daily Build Verification Test (BVT) clusters. Consequently, it is highly necessary to develop a series of tools by leveraging large-scale system tracing, big data analytics and visualization techniques to demonstrate the distributed execution of running jobs across many thousands of machines. Despite some existed works [66][67], the problem is far from settled. Furthermore, joint with techniques in software engineering of large-scale systems, research works have to be conducted to effectively improve the development and debugging of massive-scale system if the system continues to scale in the future.

History-Based Optimization (HBO) approach - Resource sharing with running workload isolation is an intuitive idea to mitigate the poor resource utilization in distributed computing system. Furthermore, accurate estimation of resource requirement could be an effective alternative. For example, for a specific compute job which daily runs in the production system, the required resource can be approximately measured and modeled considering the processed data size, paralleled instance number, operator type (e.g., some *SQL* operators such as *select, join, group, order, limit, union* and other operators such as table scan, file operations etc.). The estimated value can be further revised based on the historical resource usage of the same job type due to the assumption that the resource pattern is stable and can be followed. However, with the complexity and diversity of user-defined function (UDF) or third-party libraries and packages, the accuracy of resource estimation faces great challenges.

Simulation of large-scale system behavior - Due to the scarcity of large-scale test cluster, it is highly desirable to find a cost-effective technique to evaluate the system functionalities and performance in a simulation environment. One critical aspect of simulation is the ability to evaluate large-scale systems within a reasonable time frame while modeling complex interactions between millions of components. Additionally, the simulation approach is expected to playback the requests size and frequency in a timeline driven by high-fidelity system tracelogs.

Application in container-based system - Container-based technique has been obtaining increasing popularity recently due to the fact that it is much more light-weight compared with virtual machine. The OS-level virtualization is able to leverage the process isolation mechanism to support independent executions of co-allocated containers and the resource sharing of the same underlying resources. At present, Docker [4] rapidly achieves wide use because it can not only provide convenient and effective mechanism to deploy applications into its containers with *Dockerfiles*, but securable and isolated execution environment. Due to these reasons, the performance of typical web service composition or internet application mashup can be enhanced by using Docker. In this context, it is highly indispensable for resource management system such as [28][33][37] or specialized system such as Kubernetes [5] to provision scalable and dependable request handling, image storage, IO throughput, resource allocation in order to support large-scale container composition and orchestrations.

IoE Applications - With the booming development and the increasing demands of smart city, intelligent traffic, techniques within *Internet of Things* (*IoT*) and *Internet of Vehicles (IoV)* have become the significantly important means to realize the objectives. In addition to the hardware-related techniques such as sensor network, signal control, vehicle engineering etc., the massive-scale information system plays increasingly vital role in building effective solutions in *Internet of Everything (IoE).* There are huge demands of real-time data processing, statistical analytics and distributed machine learning in many scenarios such as user behavior pattern analysis, data mining of massive trajectory data streaming, real-time parameter tuning during unmanned automatic driving etc. Some of them are extremely safety-critical, thus have additional requirements for the dependable and real-time capability with low latency. In particular, in the architecture of "*Cloud-Network-Edge*", it is the cloud system that should be responsible for satisfying those demands above. It is noteworthy that the techniques discussed in this paper can be directly applicable within the *IoE* scenarios. Moreover, the computation resources at the edge side should also be fully utilized in tight resource environment. The executable task and process can be offloaded from the cloud side [61][61][63] to improve the holistic system utility, user QoS, and energy-efficiency.

VIII. CONCLUSIONS

In this paper we have reported our latest understanding of the main challenges in massive-scale distributed computing, and discussed both existing and potential solutions, particularly in terms of system scalability and dependability. Some important observations and conclusions can be summarized as follows:

- *Exploiting the inherent workload heterogeneity that exists in Cloud environments provides an excellent mechanism that helps to improve both the performance of running tasks and the system efficiency.* Combining specific workload types can reduce the performance degradations, limit negative effects on energy-efficiency, and improve the efficiency and effectiveness of resource scheduling.

- *Improving the scalability of a massive-scale distributed system is becoming increasingly important.* Traditional parallel processing and concurrency control techniques are often no longer suitable to a massive-scale system due to the dramatically-increased scale of its workloads and resources. Service providers have to pay a special attention to the scalability of their systems that has direct and huge economic consequences once massive and concurrent user requests cannot be handled properly.
- *Large-scale distributed systems may run millions of service instances concurrently, with an increased probability of frequent and simultaneous failures.* These failures have to be understood properly and addressed appropriately together with a correct strategy for scheduling service instances. Inappropriate scheduling of instances has the potential to dramatically affect the whole system reliability due to the complex co-relation between rescheduling and communications caused by application failures. Timing failures is also becoming an increasingly dominating failure type for modern service applications.
- *Relying on real data is critical to understanding the real challenges in massive-scale computing and formulating assumptions under realistic operational circumstances.* This is especially true in highly dynamic environments such as Cloud datacenters and big data processing systems where precise behavioral modeling is required in order to improve environmental efficiency, scalability and dependability.
- *Experiences learnt from Cloud and distributed computing will facilitate the development of the future generation computing systems that support a number of human intelligent decisions.* We believe that it is highly likely that advance in massive-scale distributed computing and big data analytics will revolutionize our way of thinking, living, and working.

ACKNOWLEDGMENTS

Special thanks must go to the SIGRS group from Beihang University, the DSS group from the University of Leeds, and the Fuxi distributed resource scheduling team in Alibaba Cloud Inc. for their support and collaborative contributions to the work discussed in this report, especially to Dr. Peter Garraghan (Leeds) and Jin Ouyang (Alibaba Cloud Inc.). The work in this paper has been supported in part by the National Basic Research Program of China (973) (No. 2014CB34-0304), China 863 program (No. 2015AA01A202), the UK EPSRC WRG platform project (No. EP/F057644/1), and Fundamental Research Funds for the Central Universities and Beijing Higher Education Young Elite Teacher Project (YETP1092).

REFERENCES

[1] http://www.cnbc.com/2015/11/10/alibaba-handles-1-billion-in-8-minutes-of-sales-through-alipay-on-singles-day.html

[2] http://www.businesswire.com/news/home/20151111006351/en/Alibaba-Group-Generated-USD-14.3-Billion-GMV

[3] ODPS: https://www.aliyun.com/product/odps/

[4] Docker Project. https://www.docker.io/, 2014.

[5] Kubernetes. http://kubernetes.io, Aug. 2014.

[6] S. Herbst-Murphy. Clearing and Settlement of Interbank Card Transactions: A MasterCard Tutorial for Federal Reserve Payments Analysts.

[7] A. McAfee and B. Erik. Big data: The management revolution. *Harvard Business Review, 10 2012.*

[8] Google Cluster Data V2 (2011). [Online] Available: http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1

[9] (2008) Amazon suffers u.s. outage on friday internet. [Online]. Available: http://news.cnet.com/

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *in Future Gener. Comput. Syst., vol. 25, pp. 599-616, 2009.*

[11] Z. Zheng, J. Zhu, and M. R. Lyu. Service-generated big data and big data-as-a-service: an overview. *In Proceedings of IEEE Big Data, 2013*

[12] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R.Das. Modeling and synthesizing task placement constraints in Google compute clusters. *In Proceedings of ACM SoCC, 2011*

[13] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. *In Proceedings of ACM SoCC, 2012*

[14] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu. An approach for characterizing workloads in Google cloud to derive realistic resource utilization models. *In Proceedings of IEEE SOSE 2013.*

[15] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu. Analysis, modeling and simulation of workload patterns in a large-scale utility cloud[J]., *IEEE Transactions on Cloud Computing, 2014*

[16] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu. An analysis of failure-related energy waste in a large-scale cloud environment, *in IEEE Transactions on Emerging Topics in Computing, 2014*

[17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *in ACM SIGOPS Operating Systems Review. ACM, 2007, 41(3).*

[18] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters [J]. *In Communications of the ACM, 2008, 51(1).*

[19] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. *In Proceedings of IEEE DSN 2004.*

[20] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. *In Proceedings of ACM SoCC, 2010, (pp. 193-204).*

[21] F. Dinu and T. Ng. Understanding the effects and implications of compute node related failures in hadoop. *In Proceedings of ACM HPDC, 2012.*

[22] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *In IEEE Transactions on Dependable and Secure Computing(TDSC),2004.*

[23] B. Randell and J. Xu, "The evolution of the recovery block concept," Software Fault Tolerance, 1995.

[24] A. Avizienis, "The methodology of n-version programming," Software fault tolerance, 1995.

[25] M. R. Lyu et al., Handbook of software reliability engineering, 1996

[26] Z. Wen, J. Cala, P. Watson, and A. Romanovsky. Cost Effective, Reliable, and Secure Workflow Deployment over Federated Clouds, *in Proceedings of IEEE Cloud, 2015*

[27] Z. Wen, J. Cala, and P. Watson. A scalable method for partitioning workflows with security requirements over federated clouds. *In Proceedings of IEEE CloudCom, 2014*

[28] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *In Proceedings of the VLDB Endowment, 2014*

[29] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: scalable and coordinated scheduling for cloud-scale computing. *In Proceedings of USENIX OSDI, 2014*

[30] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M.Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters. *In Proceedings of USENIX ATC, 2015*

[31] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. Storm@twitter. *In Proceedings of the ACM SIGMOD, 2014*

[32] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: distributed, low latency scheduling. *In Proceedings of ACM SOSP,2013*

[33] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. *In Proceedings of the USENIX NSDI, 2011*

[34] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J.Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the USENIX NSDI, 2012*

[35] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the ACM EuroSys, 2013*

[36] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. *In Proceedings of ACM EuroSys, 2015*

[37] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. "Apache hadoop yarn: Yet another resource negotiator." *In Proceedings of the ACM SoCC, 2013*.

[38] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. *In Proceedings of ACM ASPOLOS, 2014*

[39] R. Yang, T. Wo, C. Hu, J. Xu and M. Zhang. D^2PS: a Dependable Data Provisioning Service in Multi-Tenants Cloud Environments, In *Proceedings of IEEE HASE, 2016.*

[40] I. S. Moreno, R. Yang, J. Xu and T. Wo. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. *In Proceedings of the IEEE ISADS, 2013*

[41] R. Yang, I. S. Moreno, J. Xu and T. Wo. T. An analysis of performance interference effects on energy-efficiency of virtualized cloud environments. *In Proceedings of the IEEE CloudCom, 2013*

[42] Y. Wang, R. Yang, T. Wo, W. Jiang and C. Hu. Improving utilization through dynamic VM resource allocation in hybrid cloud environment. *In Proceedings of the IEEE ICPADS 2014*

[43] P. Garraghan, P. Townend and J. Xu. An empirical failure-analysis of a large-scale cloud computing environment. *In Proceedings of IEEE HASE 2014*

[44] P. Garraghan, P. Townend and J. Xu. An analysis of the server characteristics and resource utilization in google cloud. *In Proceedings of IEEE IC2E, 2013*

[45] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. MillWheel: fault-tolerant stream processing at internet scale. *In Proceedings of the VLDB Endowment, 2013*

[46] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets [J]. *In Proceedings of the VLDB Endowmen, 2010*

[47] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *In Proceedings of the VLDB Endowment, 2009*

[48] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. *In Proceedings of the ACM SIGMOD, 2010*

[49] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M.Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *In Proceedings of the VLDB Endowment 2012*

[50] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino. Apache Tez: A unifying framework for modeling and building data processing applications. *In Proceedings of ACM SIGMOD, 2015*

[51] L. Cui, J. Li, T. Wo, B. Li, R. Yang, Y. Cao and J. Huai. HotRestore: a fast restore system for virtual machine cluster. *In Proceedings of USENIX LISA, 2014*

[52] Y. Huang, R. Yang, L. Cui, T. Wo, C. Hu and B. Li. VMCSnap: Taking Snapshots of Virtual Machine Cluster with Memory Deduplication. *In Proceedings of IEEE SOSE, 2014*

[53] J. Li, J. Zheng, L. Cui and R. Yang. ConSnap: Taking continuous snapshots for running state protection of virtual machines. *In Proceedings of IEEE ICPADS, 2014*

[54] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, Design, modeling, and evaluation of a scalable multi-level check-pointing system, *In Proceedings of IEEE SC, 2010*

[55] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines." *Morgan & Claypool Publishers, 2013.*

[56] J. Dean and L. A. Barroso. The tail at scale. *In Communications of the ACM, 56(2), 2013.*

[57] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, M. Wolf, "A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-scale Datacenters", *in Proceedings of ACM ICAC, 2011*

[58] B. Mauren. Fail at scale. *In Communications of the ACM, 58(11), 2015.*

[59] R. Love, "Kernet Korner: Intro to Inotify", *Linux Journal, 139(8), 2005.*

[60] R. Ihaka,R. Gentleman,"R: a Language for Data Analysis and Graphic", Journal of Computational Graph Statistics, *1996.*

[61] Y. Zhang ,R. Yang, T. Wo, C. Hu, J. Kang and L. Cui. CloudAP: Improving the QoS of Mobile Applications with Efficient VM Migration. *In Proceedings of IEEE HPCC,* 2013

[62] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *In IEEE Pervasive Computing, 2009*

[63] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. In *Future Generation Computer Systems, 2013*

[64] X. Chen, C.-D. Lu, and K. Pattabiraman. Failure analysis of jobs in compute clouds: A google cluster case study. *In Proceedings of IEEE ISSRE, 2014*

[65] A. Rosa, L. Y. Chen, and W. Binder. Understanding the Dark Side of Big Data Clusters: an Analysis beyond Failures. *In Proceedings of IEEE DSN, 2015*

[66] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. *Technical report, Google, 2010*

[67] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *In IEEE Transactions on Parallel and Distributed Systems, 24(6), 2013*