Formal Analysis of Load Balancing in Microservices with Scenario Calculus

Hong Zhu*, Hongbo Wang†, and Ian Bayley*
*School of Eng., Comp. and Math., Oxford Brookes University, Oxford, UK, Email: (hzhu, ibayley)@brookes.ac.uk
†Dept. of Comp. Sci. & Tech., Beijing Univ. of Sci. & Tech., Beijing, China, Email: foreverwhb@ustb.edu.cn

***Abstract*—Load balancing plays a crucial role in realising the benefits of microservices, especially to achieve elastic scalability and performance optimisation. However, it is different from load balancing for virtual machines, because workloads on microservices are harder to predict and the number of services in the systems is large. In this paper, we formalise load balance as an emergent property of the microservices ecosystem, and employ scenario calculus to formally analyse the impact of scheduling on service capability and scalability. We discovered that elastic round robin scheduling is highly scalable but the service capability is limited by the slowest microservice instance. In contrast, shortest waiting queue scheduling is less scalable, but the service capability is higher.**

***Index Terms*—cloud computing; microservices; load balance; scheduling policies; scalability; service capability; scenario calculus**

I. Introduction

The microservices architecture has been widely adopted by the IT industry for cloud native applications [1]. A software application in this architecture is an ecosystem of micro-scale services [2], [3], in which, ideally, services in high demand would have more instances than those in low demand. Services running on nodes with a heavy workload would migrate to less busy nodes in order to achieve high performance. Both of these optimisations can be achieved by employing load balancers [4], [5]. They are implemented as a part of container orchestration platforms. Commercial products such as Docker Swarm, Kubernetes and NGiNX commonly provide a number of different strategies for each of these functions [6], [7]. However, their effectiveness and efficiency are not well understood. What makes load balancers more complex for the microservices is that multiple load balancers are required, each managing a large number of instances of a microservice. The workload on each microservice is far less predictable [8].

In this paper, we regard load balance as an emergent behaviour of a microservices ecosystem. We will employ scenario calculus [9] to formally analyse load balance in microservices. In order to provide guidance on the choice of load balance strategies, we study the impact of scheduling policy on service capability and scalability. For the sake of space, formal proofs of theorems are omitted.

II. Overview of Scenario Calculus

Scenario calculus is a formal method developed for reasoning about emergent behaviours in systems that consist of multiple active components, such as services [9], [10].

A. Microservices

A cloud native application in microservices architecture consists of a number $k > 0$ of services $C_1, \cdots, C_k$. At each time moment t, for each service C_i, where $i = 1, \cdots, k$, there is a variable, but finite, number n_i of service instances $A_{i,1}, \cdots, A_{i,n_i}$ running in the system. The number of instances may change with time because new instances can be created and existing ones can be terminated at runtime. An instance of a service is called an "*agent*", while the abstract service is called a "*caste*", which is a template from which agents can be instantiated. If a container technology such as Docker is used for the implementation then the abstract service (caste) is packed into a container image and instances (agents) of that service are deployed as containers.

Each service C defines a set of messages $f_i(a_1, \cdots, a_n)$, $i = 1, \cdots, m_i$, that can be sent to other services. Here, f_i is the name of the message and $a_1, \cdots, a_n$ are its parameters. They can be a request for a service, or a response to a service request. When an agent sends a message f, we say it takes action f. Each service C may also define a set of state variables $\{v_i : T_i | i = 1, \cdots, l_i, l_i \geq 0\}$, where each v_i is a variable and T_i is its data type. An agent behaves according to a set of rules, which determine how to process a message, how to change its state, and when to send a message.

B. Scenarios

A scenario is a linear sequence of actions taken by agents in the system. Let $\mathbf{A}_t$ be the set of agents in the system at time t. The basic form of scenarios is $[A_1 : \alpha_1 | p_1, \cdots, A_n : \alpha_n | p_n]$, where for each $i = 1, \cdots, n, n \geq 0, A_i \in \mathbf{A}_{t_i}$, α_i is an action that A_i performed at time t_i, p_i is a predicate on the state of agent A_i at time t_i. It represents the situation that agents $A_i (i = 1, \cdots, n, n \geq 0)$ in that order each take action α_i and in a state satisfying predicate p_i. When for all $i = 1, \cdots, n, A_i = A$, we write $A : [\alpha_1 | p_1, \cdots, \alpha_n | p_n]$ as an abbreviation. When p_i is $true$, we simply omit the predicate. We also write $A : |p$ to represent the assertion that the state of A satisfies predicate p.

Scenarios can be combined together by using logic connections $\wedge$, $\vee$ and $\neg$, and qualifiers $\forall x \in C.S$ and $\exists x \in C.S$, where S is a scenario, C is a caste, x is a free agent variable in S. They can also be combined with predicates about agents, such as an equality $x = A$ for agent variable x and agent A, a set membership $A \in C$, etc. When at a time t in an execution

of a system π, a scenario is evaluated to true, we say that system π is in scenario S at time t, and write $\pi \models_t S$.

C. Modelling Behaviours by Collections of Scenarios

The dynamic behavior of a system can be modelled by a collection of scenarios that are *complete* (covering all possible states of the system) and *orthogonal* (disjoint, thus the system can be in only one scenario at any time moment). There may be many such models, which can be composed through Cartesian product of the models as follows.

Theorem 1: Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be two complete and orthogonal sets of scenarios for a system π. The set $\mathbf{S}_1 \times \mathbf{S}_2 = \{s_1 \wedge s_2 | s_1 \in \mathbf{S}_1, s_2 \in \mathbf{S}_2\}$ is also a complete and orthogonal set of scenarios for system π. □

Let S be a scenario in $\mathbf{S}$ for a given system π. It is an *initial scenario*, written $\pi \models_0 S$, if π can be in the scenario at start time. A binary relation $\rightarrow$ on $\mathbf{S}$ is a scenario *transition relation*, if for all S and S' in $\mathbf{S}$, $S \rightarrow S'$ implies that the system π can evolve from a state in S to a state in S'. S is *reachable*, written $\pi \rightsquigarrow S$, iff there exists an initial scenario S_0 in $\mathbf{S}$ such that $S_0 \rightarrow S$. The system π *always reaches* S, written $\pi \rightarrowtail S$, if and only if for every initial state and every execution of the system, the system will reach S. Scenario S is *stable* for system π, written $\pi @ S$, if and only if for all executions of the system, the system will remain in S once it is reached. The system π *converges* to S, written $\pi \looparrowright S$, if and only if S is always reachable and is stable, i.e. $\pi \rightarrowtail S$ and $\pi @ S$.

Theorem 2 below states that removing non-reachable scenarios from a complete and orthogonal set of scenarios will affect neither the validity of the model, nor the reachability, stability and convergence of any scenario.

Theorem 2: Let $P(x)$ be a predicate on scenarios in $\mathbf{S}$ such that $\forall x \in \mathbf{S}.((\pi \rightsquigarrow x) \Rightarrow P(x))$. Then, we have that

1) $\mathbf{S}_P = \{x \in \mathbf{S} | P(x)\}$ is also a complete and orthogonal set of scenarios of π.
2) For any given scenario $x \in \mathbf{S}$, any of the statements $(\pi \rightsquigarrow x)$, $(\pi \rightarrowtail x)$, $(\pi @ x)$ and $(\pi \looparrowright x)$ is true in $\mathbf{S}$, if and only if the corresponding statement is true in $\mathbf{S}_P$.

□

III. Analysis of Load Balancing Algorithms

A. Scheduling Strategy and Service Processing Capability

We consider task scheduling first without horizontal scaling, which is added in the next subsection.

1) General Model of Load Balancing: In general, a microservices system consists of many services, each with several instances running and with several load balancers managing these instances. We focus on the simple case of one service with several instances managed by one load balancer. The results below can be generalized. The service receives requests from several sources, called service requestors below, and the load balancer distributes these requests to the pool of service instances, which are called workers. The following sets of scenarios model the service requestors, load balancer and workers, respectively.

$$\begin{aligned} Req(n) &\triangleq \{[R_1 : reqServ(r_1), \cdots, R_n : reqServ(r_n)]\} \\ LB(n) &\triangleq \{LB : [alJob(w_1, r_1), \cdots, alJob(w_n, r_n)]\} \\ W_m(n) &\triangleq \{w_m : [jobDone(r_{i_1}), , jobDone(r_{i_n})]\} \end{aligned}$$

where $R_1, \cdots, R_n$ are requestors, $w_1, \cdots, w_n$ are workers, $r_1, \cdots, r_n$ are requests, $reqServ(r)$ is the requestor's action of submitting a service request r to the system, $alJob(w_i, r_i)$ is the load balancer's action of assigning a request r_i to worker w_i for processing, $jobDone(r)$ is the worker's action of completing the service request r.

It is easy to see that $\mathbf{Req} \triangleq \bigcup_{n=0}^{\infty} Req(n)$, $\mathbf{LB} \triangleq \bigcup_{n=0}^{\infty} LB(n)$ and $\mathbf{W} \triangleq \bigwedge_{m=1}^{\infty}(\bigcup_{n=0}^{\infty} W_m(n))$ are all complete and orthogonal. Thus, from Theorem 1, we have:

Theorem 3: $\mathbf{Sc} \triangleq \mathbf{Req} \times \mathbf{LB} \times \mathbf{W}$ is complete and orthogonal. Every scenario $S \in \mathbf{Sc}$ is in the form of

$$Req(n) \wedge LB(m) \wedge W_1(d_1) \wedge \cdots \wedge W_k(d_k)$$

for some $n, m \geq 0$, $k > 0$, $d_i \geq 0$, which is denoted by $Sc(n, m, \langle d_1, \cdots d_k \rangle)$ in the sequel. □

We say that an execution of the system has a clean start with k workers, if the initial state of the system is $Init(k) = Sc(0, 0, \underbrace{\langle 0, \cdots, 0 \rangle}_{k})$, where $k > 0$.

Let $Normal^k(S)$ denote the condition on scenarios S such that it is reachable from a clean start with $k > 0$ workers, each request is assigned to one and only one worker on a first-come/first-served basis, and every worker only processes the service requests assigned to it, also in a first-come/first-served manner. Its formal definition is omitted for the sake of space. Then, by Theorem 2, we have:

Theorem 4: $\overrightarrow{\mathbf{Sc}}^k \triangleq \{S | S \in \mathbf{Sc} \wedge Normal^k(S)\}$ is complete and orthogonal for a system that has a clean start with k workers. □

2) Models of Scheduling Strategies: In $\overrightarrow{\mathbf{Sc}}^k$, only a subset of scenarios are actually reachable. These depend on the choice of scheduling strategy. We will compare two policies, round-robin and shortest waiting queue, and define the reachable subset of $\overrightarrow{\mathbf{Sc}}^k$ for each of them based on how requests are assigned to the workers. Let $w_1, \cdots, w_n$ be the sequence of workers that the load balancer assigns the requests to.

For the round-robin policy, requests $r_1, r_2, \cdots, r_k$ are assigned to worker $0, 1, \cdots, k-1$ and then request r_{k+1} is assigned to worker 0, again, and so on. Thus, if the system is in scenario $S = Sc(n, m, \langle d_1, \cdots, d_k \rangle) \in \overrightarrow{\mathbf{Sc}}^k$, we have $w_0 = 0$, $w_{i+1} = w_{i+1} \ mod \ k_i$, where k_i is the number of workers when the action $alJob(w_i, r_i)$ takes place, and $k_0 = k$. Let $RoundRobin^k(S)$ denote these conditions, and $\overrightarrow{\mathbf{Sc}}^k_{RR} \triangleq \{S \in \overrightarrow{\mathbf{Sc}}^k | RoundRobin^k(S)\}$. Then, we have the following.

Theorem 5: $\overrightarrow{\mathbf{Sc}}^k_{RR}$ is complete and orthogonal for a microservices system in which the load balancer employs round-robin scheduling and has a clean start with k workers. □

For the shortest waiting queue policy, w_i is the worker whose waiting queue is minimal among all workers at the time when the action $alJob(w_i, r_i)$ takes place. Let $QLength_t(w)$ be the length of the waiting queue for worker w at time t. Let $ShortestQueue^k(S)$ be a predicate that for all $S = Sc(n, m, \langle d_1, \cdots, d_k\rangle) \in \overrightarrow{\mathbf{Sc}}^k$, the predicate is true, if and only if for all $i = 1, \cdots, m$, $QLength_t(w_i) \leq QLength_t(w)$ for all workers w in the system at the time when the action $alJob(w_i, r_i)$ takes place. Let $\overrightarrow{\mathbf{Sc}}^k_{LW} \triangleq \{S \in \overrightarrow{\mathbf{Sc}}^k | ShortestQueue^k(S)\}$. Then, we have:

Theorem 6: $\overrightarrow{\mathbf{Sc}}^k_{LW}$ is complete and orthogonal for a microservices system that employs the shortest waiting queue scheduling policy and has a clean start with k workers. □

3) Analysis of Service Processing Capability: The capability of a service-oriented system is the maximal service request rate at which the number of service requests waiting for processing will not increase indefinitely. It is determined by three factors: (a) the throughput of the individual workers, (b) how requests are distributed to them, and (c) the throughput of the load balancer itself.

Let $\delta(k)$ denote the load balancer's throughput for k workers, i.e. the maximal number of requests that the load balancer can assign to k workers per second. The length of waiting queue for a load balancer to distribute the service requests to k workers will increase indefinitely if the service request rate ρ is greater than the throughput $\delta(k)$ of the load balancer, i.e. when $\rho > \delta(k)$.

At time t, the length of waiting queue for worker w depends on not only the scheduling strategy but also the worker's throughput, i.e. the number of requests that can be handled per second, which we assume is constant for a given worker.

For the shortest waiting queue policy, the waiting queue for the system as a whole will increase indefinitely only when all the individual workers' waiting queues do so. Thus, the system's capability is $Min(\delta(k), \sum_{i=1}^{k} \theta_i)$, assuming that the system has k workers $w_1, \cdots, w_k$ whose throughputs are $\theta_i, i = 1, \cdots, k$. Note that $\sum_{i=1}^{k} \theta_i$ is the maximal possible capability of a set of workers.

For the round-robin policy, we have that the length of the waiting queue for a worker with throughput θ will increase indefinitely iff $\rho > k \cdot \theta$. The total length of the waiting queues for the whole system will increase indefinitely if one of the workers do so, i.e. iff $\rho > k \cdot \theta_{min}$, which is the minimum throughput of the workers.

Consequently, in a load balanced system with round robin scheduling policy, and a variable service request rate $\rho(t)$ with $\rho(t) < k \cdot \theta_{min}$ at all times t, then the waiting queue length for any worker remains no more than one at any time moment, provided that the system had a clean start. This means that we can always reach a stable scenario in which every worker has a waiting queue length of zero (in which the worker is idle) or one (in which no more requests have been assigned to it) provided that the service request rate does not exceed the system's capability. Moreover, the system will return to this scenario if a surge in service request rate takes it away from that. In other words, the capability of the microservice system using round-robin scheduling policy equals $Min(\delta, k \cdot \theta_{min})$. If all workers are of the same throughput θ, the system's capability is $Min(\delta, k \cdot \theta)$.

B. Capability Management and Horizontal Scalability

The analysis in the previous subsection reveals that a system's capability is affected by the number k of workers and their throughputs θ_i. In practice, the load balancer changes the values of k and/or θ'_i to new values k' and θ'_i to ensure that the system's capability stays above ρ. Running a service on a more powerful node so that θ_i increases is known as vertical scaling. Varying k is known as horizontal scaling and is the subject of this section.

1) Model of Horizontal Scaling: A load balancer performs two actions to achieve horizontal scaling:

1) $addWorker()$, which creates a new worker and adds it to the set of service provider instances so that jobs can be allocated to it;
2) $removeWorker()$, which terminates a worker and removes it from the set of service provider instances so that no more jobs can be allocated to it.

The set of scenarios for an elastic load balancer that has added a workers, removed r workers and allocated m jobs is given by the following.

$$LB_E(a, r, m) \triangleq \{LB : [\chi_1, \cdots, \chi_n]\}$$

where $n = a+r+m$, and for each $i = 1, \cdots, n$, χ_i is an action of $alJob(w, r)$, $addWorker()$ or $removeWorker()$. It is easy to see that $\mathbf{LB}_E \triangleq \bigcup_{a,r,m\geq 0} LB_E(a, r, m)$ is orthogonal and complete for any elastic horizontal scaling load balancer.

In such a system, the behaviours of workers and service requesters are exactly the same as before. Thus, $\mathbf{Sc}_E \triangleq \mathbf{Req} \times \mathbf{LB_E} \times \mathbf{W}$ is orthogonal and complete for the elastic load balancing system. Again, the elements in $\mathbf{Sc}_E$ are in the form of $Req(n) \wedge LB_E(a, r, m) \wedge W_1(d_1) \wedge \cdots \wedge W_k(d_k)$, which we shall write as $Sc_E(n, \langle a, r, m\rangle, \langle d_1, \cdots, d_k\rangle)$. Let $Elastic^K(S)$ be the predicate on scenarios of $\mathbf{Sc}_E$ such that for all scenarios $S = Sc_E(n, \langle a, r, m\rangle, \langle d_1, \cdots, d_k\rangle) \in \mathbf{Sc}_E$, the predicate is true, if $k = K + a - r$, and if the last action of the load balancer in scenario S is $addWorker()$ or $removeWorker()$, then $d_k = 0$, and the waiting queue length for worker k is 0. Let $\overrightarrow{\mathbf{Sc}}^k_E = \{S \in \mathbf{Sc}_E | Elastic^k(S)\}$. Then, $\overrightarrow{\mathbf{Sc}}^k_E$ is complete and orthogonal for normal microservices systems with horizontal elastic scaling that have a clean start with k workers.

It is worth noting that the definitions of predicates $RoundRobin(S)$, $ShortestQueue(S)$ given in the previous subsection can be easily generalized to $\overrightarrow{\mathbf{Sc}}^k_E$ for elastic scheduling. The analysis given in the previous section also holds for systems with horizontal elastic scaling. Details are omitted for the sake of space.

2) Analysis of Horizontal Scalability: A key question about load balancing algorithms is whether there is an upper limit in the number of workers above which adding more workers will

not improve the systems capability. We call this upper limit the scalability of the system.

Let δ be the throughput of the load balancer, and $\theta_{min} > 0$ be the minimal throughput of the workers in the system. We have the following.

Theorem 7: For round robin scheduling, the scalability is given by $K_{max} = \lceil \rho/\theta_{min} \rceil$. □

In particular, when all workers have the same throughput θ, then the scalability of a system that uses round-robin scheduling policy is $\lceil \delta/\theta \rceil$.

For shortest queue scheduling, in contrast, the selection of a worker takes time at least $O(log\ k)$. Even for small values of k, its throughput is smaller than that of round robin.

Theorem 8: Let $0 < \theta_1 \leq \cdots \leq \theta_n \leq \cdots$ be the throughputs of the workers in the system. For the shortest queue scheduling policy, the system's scalability is upper bounded by $K_{max} = Max_k(Min\{\delta(k), \sum_{i=1}^{k} \theta_i\})$. □

In particular, if all workers in the system have the same throughput θ, and assuming the scheduling policy has a computational complexity of $C \cdot log(k)$, then the scalability of the system is the value k such that $k/log(k) = C/\theta$.

From the above, we can conclude that round robin is more horizontally scalable than the shortest queue policy.

IV. Conclusion

A. Related Work

Load balancing has long been studied intensively, originally for distributed computing [11] and more recently for cloud computing, leading to a large number of proposed scheduling strategies and techniques; see [12], [13] for surveys. Existing work can be classified into two categories: (a) devising workload models for task scheduling and service capability management, and (b) allocating virtual machines to physical machines to optimise performance and resource usage. As Fazio et al. pointed out [7], it is difficult to build workload prediction models for microservices. It is also infeasible to use them because each application may have thousands of microservices. Monitoring a large number of microservices will cause an overhead too high to be practical. Theoretically speaking, the works on virtual machine allocation and migration are applicable to container/microservices allocation and migration. But, again, it will be too costly to perform optimisation each time a microservice is created, since the creation and termination of microservices are much more frequent than virtual machines. After all, for microservices architecture, scalability is a main concern, which has not been addressed in existing work on load balance.

B. Main Contributions of the Paper

In this paper, we presented a formal analysis of load balancing algorithms in microservices with focus on the impact of scheduling policies on service capability and scalability. First, we proved that round-robin scheduling policy has better scalability than the workload aware policy of shortest waiting queue, while the latter achieves a higher processing capability. These results have not been reported in the literature as far as we know.

Secondly, we have used scenario calculus to analyse the properties of microservices systems, which is a novel approach for studying distributed systems. In this approach, we regard a microservices system as an ecosystem in which a large number of instances of microservices are dynamically created and terminated. They communicate and collaborate with each other to achieve designed emergent properties and demonstrate emergent behaviour. Scenarios and the transition relations between the scenarios provides a model of the system's behaviour at a very high level of abstraction. This enables us to deal with the high complexity of the dynamic concurrent executions of the system. This paper demonstrates that the use of scenario calculus for microservices is feasible. Existing formal methods, such as Petri-nets, process algebra, labeled transition systems, etc., cannot be applied to systems of microservices due to its dynamic ecosystem features. Existing evaluations of load balancing algorithms have been empirical and/or simulation-based. No formal method has been applied to such analyses as far as we know.

C. Future work

We are also applying the scenario calculus to other strategies of load balancing. It is worth noting that load balancing can be combined with fault-tolerance techniques in cloud computing. This is worth further investigation.

References

[1] Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J. and Josuttis, N., *Microservices in Practice*, Part 1/2. IEEE Software, Vol. 34, No.1/2, pp91-98/97-104, Jan./ Mar., 2017.

[2] Lewis, J., and Fowler, M., *Microservices*. URL: http://martinfowler.com/articles/microservices.html#footnote-monolith, 25 Mar. 2014. (Last access on 2 Nov. 2015)

[3] Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M., *Microservice Architecture: Aligning Principles, Practices, And Culture*. O'Reilly Media, Inc., June 2016.

[4] Krause, L., *Microservices: Patterns and Applications*. Amazon.co.uk, April, 2015.

[5] NewMan, S., *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, Feb., 2015.

[6] Khan, A., *Key Characteristics of a Container Orchestration Platform to Enable a Modern Application*, IEEE Cloud Computing, Vol. 4, No. 5, pp. 42-48, Sept./Oct. 2017.

[7] Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. and Merle, P., *Elasticity in Cloud Computing: State of the Art and Research Challenges*, IEEE Transactions on Services Computing, DOI 10.1109/TSC.2017.2711009.

[8] Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L. and Villari, M., *Open Issues in Scheduling Microservices in the Cloud*, IEEE Cloud Computing, Vol.3, No.5, pp81-88, Sept. 2016.

[9] Zhu, H., *Formal Reasoning about Emergent Behaviour in MAS*, Proc. of SEKE'05, pp280-285, July 2005.

[10] Zhu, H., Wang, F. and Wang, S. *On the Convergence of Autonomous Agent Communities*, Multi-Agent and Grid Systems, Vol. 6. No. 4, pp315-352, Dec. 2010.

[11] Andrews, G. R., Dobkin, D. P., and Downey, P. J., *Distributed Allocation with Pools of Servers*, in Proc. of PODC '82, pp73-83, 1982.

[12] Nuaimi, K. A., Mohamed, N., Nuaimi, M. A. and Al-Jaroodi, J., *A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms*, in Proc. of NCCA 2012, pp137-142, 2012.

[13] Shaw, S. B. and Singh, A. K., *A survey on scheduling and load balancing techniques in cloud computing environment*, in Proc. of ICCCT 2014, pp87-95, 2014.