Front. Eng. Manag. 2025, 12(2): 272–290
https://doi.org/10.1007/s42524-023-0272-2

RESEARCH ARTICLE

Shuyi MA, Jin LI, Jianping LI, Min XIE

Cloud-integrated cyber–physical systems: Reliability, performance and power consumption with shared-servers and parallelized services

Abstract Cloud systems, which are typical cyber–physical systems, consist of physical nodes and virtualized facilities that collaborate to fulfill cloud computing services. The advent of virtualization technology engenders resource sharing and service parallelism in cloud services, introducing novel challenges to system modeling. In this study, we construct a systematic model that concurrently evaluates system reliability, performance, and power consumption (PC) while delineating cloud service disruptions arising from random hardware and software failures. Initially, we depict system states using a birth–death process that accommodates resource sharing and service parallelism. Given the relatively concise service duration and regular failure distributions, we employ transient-state transition probabilities instead of steady-state analysis. The birth–death process effectively links system reliability, performance, and PC through service durations governed by service assignment decisions and failure/repair distributions. Subsequently, we devise a multistage sample path randomization method to estimate system metrics and other factors related to service availability. The findings highlight that the trade-off between performance and PC, under the umbrella of reliability guarantees, hinges on the equilibrium between service duration and unit power. To further delve into the subject, we formulate optimization models for service assignment and juxtapose optimal decisions under varying availability scenarios, workload levels, and service attributes. Numerical results indicate that service parallelism can improve performance and conserve energy when the workload remains moderate. However, as the workload escalates, the repercussions of resource sharing-induced performance loss become more pronounced due to resource capacity limitations. In cases where system availability is constrained, resource sharing should be approached cautiously to ensure adherence to deadline requirements. This study theoretically analyzes the interrelations among system reliability, performance, and PC, offering valuable insights for making informed decisions in cloud service assignments.

Keywords cloud service modeling, transient downtime analysis, resource sharing, service parallelism

Received Apr. 30, 2023; revised Jul. 31, 2023; accepted Aug. 28, 2023

Shuyi MA
School of Management, Xi'an Jiaotong University, Xi'an 710049, China; Department of Systems Engineering, City University of Hong Kong, Hong Kong, China

Jin LI (✉)
School of Management, Xi'an Jiaotong University, Xi'an 710049, China
E-mail: jinlimis@xjtu.edu.cn

Jianping LI
School of Economics and Management, University of Chinese Academy of Sciences, Beijing 100190, China

Min XIE
Department of Systems Engineering, City University of Hong Kong, Hong Kong, China

This research was supported by the National Natural Science Foundation of China (Grant Nos. 72372131, T2293774, and 71901169), the Shaanxi Province Innovative Talents Promotion Plan – Youth Science and Technology Nova Project (Grant No. 2022KJXX-50), and the Youth Talent Promotion Project of China Association for Science and Technology (Grant No. YESS20200072).

1 Introduction

Cloud computing services have been evolving rapidly in response to the flourishing digital economy. Infrastructure-as-a-service (IaaS) stands out as a prominent application widely embraced by individuals, governments, and enterprises (N'Takpé et al., 2022). Under the IaaS paradigm, cloud service users access computing resources at the information technology infrastructure level on demand and remunerate based on their usage volume. Cloud service providers (CSPs) are accountable

for system maintenance and request scheduling. Quality of service (QoS) attributes, encompassing availability and response time, are stipulated through service level agreements (SLAs) between CSPs and users. Failure to attain QoS objectives subjects the CSP to penalty costs, leading to a tarnished reputation, and necessitates compensation for users according to the SLAs. CSP administrators often juggle multiple SLAs for different users while considering supplementary operational costs such as electricity expenses, which can amount to as much as 70% of the daily operational outlays of cloud computing data centers (Zaloumis, 2022). As a result, effectively harnessing existing computing resources to ensure service quality within controlled operational expenses persists as a challenge confronting CSPs in scheduling cloud services.

Despite the ongoing global expansion of the cloud data center market, statistics reveal that the global average resource utilization rate of data centers hovers at merely approximately 10%–20% (Guo et al., 2019a). Virtualization technology emerges as a pivotal solution for markedly enhancing resource utilization. Within the framework of virtualization technology, CSPs can transform a physical node into multiple virtualized components, namely, virtual machines (VMs) or containers. CSPs can regard these abstract virtual facilities as basic units of service orientation (Li et al., 2023). Colocated services partake in the physical resources of a node but remain segregated via distinct mappings between software and hardware. Cloud environments frequently adopt parallel architectures. The majority of cloud computing services are designed to be malleable, enabling them to operate concurrently across physical nodes (Gupta et al., 2014; Harchol-Balter, 2021). Cloud services can be fractionated into multiple subservices and deployed on diverse nodes to expedite service delivery. Virtualized resources and multiple cloud services distributed across physical nodes collectively constitute a typical cyber–physical system (CPS) (Guo et al., 2023b). The tangible hardware manages digital services, while the virtualized software facilitates their execution. The interactions among potentially colocated services and parallelized subservices significantly influence system performance. Resource sharing and service parallelism confer greater flexibility in service allocation, and their ramifications on system metrics remain a subject necessitating further analysis.

Preexisting studies have formulated models for cloud service assignments and proposed optimal strategies grounded in various metrics. Prominent metrics frequently examined encompass service performance, power consumption (PC), and system reliability. However, few studies have considered the intricate interdependencies among these metrics, potentially stemming from dissimilar definitions of performance and reliability. This divergence results in a lack of comparable links between diverse aspects (Qiu et al., 2016). In this study, our focus revolves around the predicament of service assignment within a cloud environment, where random hardware and software failures persist in the milieu of parallel services and shared resources. Our approach centers on considering reliability, performance, and PC stemming from the execution process of services, rather than concentrating solely on hardware configuration, backup settings, or service queues. The ensuing paragraphs provide a succinct overview of these three dimensions. Reliability control seeks to minimize service downtime caused by failures, leading us to gauge reliability based on service duration. Given that services have deadline requirements, we employ the proportion of services completed within stipulated timeframes as a measure of system reliability. Furthermore, performance can be succinctly summarized as system throughput, with the aggregate duration of all services representing system performance. For system power consumption, we summate the PC of each physical node, computed as the product of unit power and active time.

This study centers on cloud-integrated CPSs within the realm of resource sharing and service parallelism. The research questions are outlined as follows: 1) How can the assessment of system reliability, performance, and PC be conducted? 2) In what manner do resource sharing and service parallelism influence system metrics? And 3) how can service assignments be optimized to attain equilibrium between performance and PC while adhering to system reliability constraints? The birth–death process has been widely employed for modeling stochastic system failures and repairs. Numerous preceding investigations have undertaken analyses under steady-state conditions to quantify the values of system metrics. Nonetheless, some studies have ascertained that within a limited service window, completion of the service might occur before reaching a steady state (Du et al., 2015; Fahmideh et al., 2019; Guo et al., 2020). Addressing research question 1), we leverage transient probabilities to model the birth–death processes of services and extend the groundwork of Du et al. (2015) to estimate downtime distributions encompassing fragmented service windows. Our proposed model interconnects system metrics through service time, a derivative of service assignments and system availability. Regarding research question 2), we illustrate that service parallelism impacts the workload of each subservice, hastening its execution. However, resource sharing curtails the processing rate of nodes that concurrently handle multiple services. The interplay between service parallelism and resource sharing dictates the minimum execution time required for each subservice. Additionally, we formulate mixed-integer programming models for optimizing performance and PC, offering enhanced comprehension of research question 3). Subsequent to conducting numerical experiments, we compare the degrees of service parallelism and resource sharing within optimal assignment decisions across system

availabilities, workload levels, and service characteristics. This model can be effectively employed to optimize service assignments in practical scenarios while also facilitating the analysis of interactions among system metrics. Under the confines of reliability constraints, the disparity introduced by prioritizing performance or PC fluctuates according to workload, availability levels, and service patterns.

The principal contribution of this study resides in the formulation of a comprehensive framework for quantifying and enhancing the system reliability, performance, and energy efficiency of cloud-integrated CPSs. We theoretically elucidate the ramifications of service parallelism and resource sharing on system metrics and interdependencies. Our framework furnishes efficient evaluations of system metrics through transient properties. This underscores discrepancies between hardware and software failures and can be extended to diverse availability conditions. This study serves to assist administrators of CSPs in making informed decisions concerning service assignments, particularly within virtualized settings in cloud computing data centers. Administrators can assess the influence of various decisions on system metrics and dynamically optimize them in accordance with practical requisites.

The subsequent sections of this paper are structured as follows. Section 2 presents a review of the pertinent literature. Section 3 introduces the formulations of system metrics and models the birth–death process utilizing transient probabilities. Section 4 expounds upon algorithms for state transitions in path generation and multistage downtime estimation. Section 5 offers a comparative analysis and optimization of service management, elucidating the roles of resource sharing and service parallelism. Finally, Section 6 concludes the paper, outlining limitations and avenues for future research endeavors.

2 Related works

This study primarily intersects with three research streams: Cloud system modeling, service downtime estimation, and service assignment decisions. We provide a concise literature review and position our work as follows.

2.1 Cloud system modeling

Comprehending failure/repair distribution and the implications of failures holds paramount significance in cloud system modeling. Guo et al. (2020) employed exponential, Weibull, and Erlang distributions to represent normally functioning, aging, and partially failing systems, respectively. They devised a birth–death process for multiple VMs under a one-to-one mapping with physical nodes. Li et al. (2021) modeled service reliability incorporating resource sharing, with a focus on the influence of hardware failures. Cloud service systems possess the unique attribute of common-cause failures due to virtualization technology (Qiu et al., 2016). Colocated VMs can experience simultaneous failures due to physical node issues or remain independent of one another in cases of software failures. Qiu et al. (2019) constructed a model that integrated cloud system metrics encompassing reliability, performance, and PC while considering both hardware and software failure/repair processes. Their definition of system reliability involved downtime of physical nodes and VM failures, estimated when the system reaches steady state. The research framework of this study similarly incorporates downtime within metric formulations. However, our focus diverges from reducing downtime; instead, we explore how downtime affects system metrics and service assignment decisions. The cloud failure model by Qiu et al. (2019) considers common-cause failures with resource sharing and optimizes service assignment correspondingly. They assumed that service time is independent of allocation specifics, depending solely on service attributes. However, such an assumption might not align with broader and more practical scenarios. Han et al. (2020) modeled resource contention within virtualized cloud systems and assessed the extent of processing rate degradation through experimental testing. The resource contention of colocated virtual facilities on resources that cannot be fully isolated in physical nodes (e.g., bandwidth and memory) results in a reduced processing rate as the number of virtual facilities increases. Concurrently, service parallelism introduces variability in the service time of each subservice, influenced by its allocated workload (Feng and Huang, 2015) and the quantity of subtasks (Harchol-Balter, 2021).

The system metric represents the objective goals of different interests. Reliability often collaborates with minimized availability loss (Izrailevsky and Bell, 2018). Availability, a common stipulation in cloud SLAs, denotes the minimum uptime percentage throughout the service duration. Tian et al. (2020) devised a failure prediction framework grounded in machine learning techniques, assessing system reliability from the user's viewpoint, quantifying the probability of completing a service devoid of failures. System performance chiefly relates to service time optimization. For instance, service queues are modeled and fine-tuned to diminish service waiting times (Niño-Mora, 2019; Wang et al., 2020; Guo et al., 2022) while also optimizing the effect of the physical node processing rate on the service response time (Eshraghi and Liang, 2019; Priya et al., 2019). The former literature stream typically revolves around maximizing resource utilization within limitations, whereas the latter emphasizes distinctions in processing rates among nodes influenced by heterogeneous resource configurations or performance reductions arising from resource sharing. The power model estimates unit power

for specific system states. The linear power model, where a physical node's unit power is proportional to its utilization, garners extensive usage due to its balance between accuracy and simplicity (Lin et al., 2018).

In this context, we treat availability as an external system parameter influenced by failure and repair distributions. Rather than concentrating on fault-tolerant design, we explore other system metrics within predefined availability conditions. Hence, we regard the proportion of on-time services, total service time, and PC of physical nodes as proxies for system reliability, performance, and PC, respectively. Our approach employs the birth–death process to model service executions, accommodating random hardware and software failures/repairs, while additionally factoring in the effects of resource sharing and service parallelism on service speed.

2.2 Downtime estimation

Utilizing steady-state probabilities to infer system states presents a straightforward approach. Qiu et al. (2019) derived birth–death process state probabilities through the Chapman–Kolmogorov equation and estimated anticipated downtimes. Cotroneo et al. (2022) employed a higher-order Markov model to represent cloud system failures, estimating hardware failures via steady-state transitions and empirical simulations. In fact, during relatively brief service durations, such estimations might lack accuracy, as facilities could bypass a steady state (Du et al., 2015). Fahmideh et al. (2019) underscored that transient faults are the more probable cause of cloud service failures. Ivanchenko et al. (2021) analyzed availability loss due to transient server failures, underscoring the significance of transient analysis in device performance deterioration. Drawing from empirical analysis of historical operational data, the median duration of a national data center's service is less than one and a half hours (Guo et al., 2023a). This timeframe is notably short compared to the average time to failure in cloud systems. Du et al. (2015) introduced an efficient approach for transient probability analysis by generating ample random sample paths to empirically estimate the overall downtime distribution. Furthermore, Guo et al. (2020) effectively demonstrated the application of this approach in modeling random failures with multiple services. Hence, in this context, we propose the adoption of transient analysis over steady-state derivation, further extending the sample path randomization (SPR) technique to multistage scenarios. The algorithm we propose estimates downtime distribution stemming from hardware and software failures featuring fragmented service windows.

2.3 Service assignments

Distinct service-assignment policies yield diverse operational outcomes for the system. Service assignment poses a Non-deterministic Polynomial (NP)-hard challenge (Ibrahim et al., 2020). Wang et al. (2022) optimized service assignments to curtail instances of deadline-reliability breaches. Their optimization accounted for heterogeneous physical nodes and hardware failures, featuring an efficient problem-solving approach. Ibrahim et al. (2020) juxtaposed various service assignment strategies in cloud computing, affirming that load balancing bolsters system throughput. Guo et al. (2023a) illuminated the impact of workload distribution within cloud systems on PC and devised a decision tree empowering CSP administrators to allocate services across distinct conditions. Furthermore, while numerous studies have scrutinized single metrics, there has been ample exploration of trade-offs amidst system metrics. For instance, Garg et al. (2019) introduced a model to optimize energy efficiency and reliability via dynamic voltage and frequency scaling (DVFS) of physical nodes. However, they abstained from analyzing performance shifts due to DVFS adjustments. Ataie et al. (2022) modeled an energy-conscious central manager amalgamating service assignments and performance safeguarding. Lin et al. (2022) delved into an online VM consolidation strategy aimed at augmenting energy efficiency and performance sans violating SLAs. Qiu et al. (2019) unearthed that the crux rests on the frequency of physical nodes, optimizing performance and PC through processing frequency modulation. Notably, these studies stand as technically sound contributions, yet they lack theoretical trade-off analyses. Our study endeavors to contrast the PC and performance trade-offs across various downtime settings and system workload intensities. We discern that potential gains and losses may manifest distinctively across divergent scenarios.

3 System modeling

In this section, we introduce the mechanisms of resource sharing and service parallelism within cloud-integrated CPSs. Subsequently, we model the process of service execution within a single physical node to distinctly elucidate the disparities between hardware and software failures/reparations. Moving forward, we illustrate the birth–death process inherent in a system featuring multiple services and nodes while expounding upon system evaluations. The primary symbols and pertinent explanations are succinctly summarized in Table A1 in Appendix A in the Supplementary Material.

3.1 Resource sharing and service parallelism

The resource requirements of cloud service users exhibit heterogeneity, potentially varying significantly based on their distinct computational demands. Nevertheless, the hardware configurations of physical nodes in a cloud data

center tend to be constrained (Liang et al., 2021). In practice, diverse and fragmented service requests may not be accommodated on a singular node. Utilizing service parallelism and resource sharing within an adaptable cloud framework becomes paramount to effectively allocate services.

Parallel architectures, exemplified by mechanisms such as message passing interface (MPI) programing or the MapReduce paradigm, facilitate the segmentation of a service into multiple subservices. These subservices operate concurrently across different nodes within the system (Li et al., 2021; Wang et al., 2021a). Adhering to Amdahl's law, parallel computing utilizing multiple processors can expedite task execution. Amdahl (1967) introduced the concept of the speedup ratio, which signifies the ratio between execution time using a single processor versus execution time using multiple processors. Equation (1) encapsulates the theoretical speedup ratio, contingent upon the variables f and m. Here, f denotes the fraction of execution time that benefits from parallelism, while m represents the number of assigned processors.

$$Speedup = \frac{ExecutionTime_{\text{single processor}}}{ExecutionTime_{m\ \text{processors}}} = \frac{1}{(1-f)+f/m}. \quad (1)$$

Amdahl's law, when applied within the realm of multiprocessors, can also be extended to encompass multiserver systems within modern cloud-integrated CPS. In this context, envision a scenario where a task necessitates one unit of time for execution on a singular node. When this task is executed in parallel across m nodes, its time savings can be quantified using a speedup function. This function is typically represented as a power function of m, as indicated in Eq. (2). The scalability factor $0 \leqslant \rho \leqslant 1$ embodies the extent to which a service can be accelerated, contingent upon its specific design (Madni et al., 2017; Harchol-Balter, 2021). A larger value of ρ corresponds to greater parallelism benefits for the service. In this context, we introduce the concept of a one-service-multiple-node configuration for service parallelism.

$$Speedup(m) = \frac{ExecutionTime_{\text{single node}}}{ExecutionTime_{m\ \text{nodes}}} = m^{\rho}. \quad (2)$$

In a similar vein, virtualization technology facilitates resource pooling and the on-demand creation of desired virtual configurations. While these colocated virtual facilities, which are mapped to the same physical node, operate independently, they still necessitate hardware resources (Cao et al., 2023), including bandwidth or memory. Resource contention within these scenarios can lead to a reduction in the processing rate of the associated nodes. The extent of this speed degradation is influenced by factors such as the number of colocated services and their patterns of resource consumption (Canosa-Reyes et al., 2022). Similar to the speedup function utilized in service parallelism, studies have demonstrated that the average speed degradation can be estimated using a power function of the count of colocated services (Bai et al., 2011; Zhang et al., 2014; Islam et al., 2017). As depicted in Eq. (3):

$$SpeedDegradation(n) = \frac{Throughput_{n\ \text{services}}}{Throughput_{\text{single service}}} = n^{-\alpha}, \quad (3)$$

where n is the count of colocated services, and $0 \leqslant \alpha \leqslant 1$ represents the coherency factor associated with these services. A greater value of α corresponds to a heightened resemblance in the resource usage patterns among services, consequently intensifying resource contention on the node. In this context, we introduce the concept of a one-node-multiple-service framework to elucidate resource sharing dynamics.

3.2 Service execution process in one node

The availability of the system hinges upon the occurrence patterns of failures and repairs during the execution of cloud services. Here, two categories of random failures are considered: Hardware failures and software failures. Hardware failures often stem from unstable power supplies, aging equipment, and disk damage, whereas software failures primarily arise from configuration errors and kernel crashes. To avert such failures during service execution, it is imperative for CSPs to conduct targeted repairs both efficiently and promptly.

For in-depth analysis, it is crucial to establish a clear comprehension of the distribution of failures and repairs. Typically, CSPs can derive estimates for the mean time to failure and mean time to repair for both physical nodes and virtual facilities through the empirical analysis of historical operational data (Sharma et al., 2019). This study assumes that the patterns of failure and repair can be projected based on historical operational data. Notably, the consequences stemming from hardware and software failures exhibit differences. In cases of software failures, where virtual facilities are isolated and independent, such failures exclusively affect the associated virtual facility (Qiu et al., 2019; Sayadnavard et al., 2019). Conversely, due to the resource sharing nature of physical nodes, hardware failures impact all virtual facilities hosted on them (Qiu et al., 2019; Wang et al., 2021b). We denote the rates of hardware and software failures as λ_h and λ_s, respectively, representing the reciprocals of their respective mean times to failure. The approach to recovery also varies between hardware and software repairs. Common fault-tolerant designs for cloud services include checkpointing strategies and autoretry mechanisms (Setlur et al., 2020; Levitin et al., 2023). The assumption here is that services are backed up in real time prior to failure incidents, and they can automatically resume execution post-repair. In cases of software failures, recovery involves minimal rollback to its backup, enabling the

service to immediately resume its execution progress (Chinnathambi et al., 2019; Malik et al., 2022). On the other hand, when hardware failures occur, the backup for all colocated services is lost. Consequently, all virtual facilities revert to their initial state, necessitating the restart of all services from the beginning (Zhang et al., 2021). The repair rates for hardware and software failures are denoted as μ_h and μ_s, respectively, and can be empirically determined based on the reciprocals of their respective mean times to repair.

Figure 1 illustrates the service execution process within a single node. The states of regular service operation, software fault interruption, hardware fault interruption, and service completion are represented by 1, 0, −1, and 2, respectively. Completion of a service (i.e., transitioning to state 2) can only occur if it remains in state 1 (regular operation) for a duration that satisfies the required workload. Denoted as T^w, this duration represents the minimum time necessary to fulfill the service's workload requirements. If a hardware failure arises after a time interval t_{fh} (the transition from state 1 to state −1), the progress on this node is reset. Following a repair time interval t_{rh} (transition from state −1 to state 1), the service recommences. Similarly, suppose a software failure occurs after a time interval t_{fs} (transition from state 1 to state 0). In that case, the service remains in a preinterruption stage unless a subsequent hardware failure occurs (transition from state 0 to state −1). After a time interval t_{rs} for software repair (transition from state 0 to state 1), the service's progress persists until it reaches completion (transition from state 1 to state 2). The autoretry mechanism facilitates the automatic restart of services post-failure without requiring additional reconfiguration. Each restart process is executed independently, and the number of hardware failures (l − 1, for instance) conforms to a geometric distribution. The random variables t_{fh}, t_{fs}, t_{rh}, and t_{rs} adhere to specific distributions with parameters λ_h, λ_s, μ_h, and μ_s, respectively. The minimum operational time requirement, T^w, is a deterministic variable contingent upon workload requisites and the speeds of resource processes.

3.3 Birth–death process for multiple nodes

We consider a cloud-integrated CPS comprising a set of $|N|$ homogenous physical nodes $N = \{n_1, n_2, \ldots, n_j, \ldots, n_{|N|}\}$, and a set of $|S_t|$ ongoing services $S_t = \{s_1, s_2, \ldots, s_i, \ldots, s_{|S_t|}\}$ at time t. We use s_{ij} to represent the subservices of s_i executing on node n_j. To delineate the degree of resource sharing and parallelism, we introduce a binary variable a_{ij} to indicate the presence of s_{ij}, with $a_{ij} = 1$ signifying the assignment of service s_i to node n_j and 0 otherwise. Thus, at any given time t, there exist $x_{jt} = \sum_{i=1}^{|S_t|} a_{ij}$ uncompleted services operating on node n_j and $m_t = \sum_{j=1}^{|N|} \min\{1, x_{jt}\}$ active physical nodes. As service completions lead to resource releases in a stochastic manner, the values of m_t and x_{jt} also dynamically shift along the temporal axis. Considering that the number of running services and active nodes remain constant within the time interval following the first service completion, we can simplify the notation within this service window by omitting the time subscripts on x_{jt} and m_t and substituting them with x_j and m in the model. Meanwhile, the binary variables h_j and b_{ij} are utilized to indicate whether node n_j is under hardware failure conditions and whether subservice s_{ij} is incapacitated due to software failure, respectively.

Figure 2 illustrates the birth–death process for a system featuring multiple nodes and several concurrently operating services. Each state is represented across multiple nodes and elaborated as follows: On node n_j, there are a maximum of x_j services in the uptime state (state 1). In the event of a random software failure occurring on node n_j, one service transitions from state 1 to state 0, resulting in (x_j − 1) services in the uptime state. If a random hardware failure affects node n_j, we utilize −1 (all x_j services shift to state −1) to distinguish this scenario from state 0, where x_j instances of software failures would occur.

In conventional methodologies, steady-state analysis is commonly employed for the formulation of transition probabilities. Nonetheless, the service period is generally short, and the system might not attain a steady state prior to the completion of the service. Drawing inspiration from the work of Du et al. (2015) and Guo et al. (2020), we suggest employing transient probabilities to capture state transitions. Let us consider a scenario where a system is in a state featuring k operational nodes and z uptime subservices. When the duration of the sojourn interval, denoted as Δt, approaches a minuscule value near zero, the state transition occurring within this Δt is

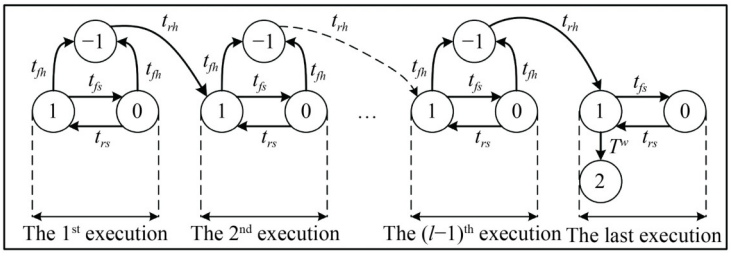

Fig. 1 Illustrations of service execution state transitions.

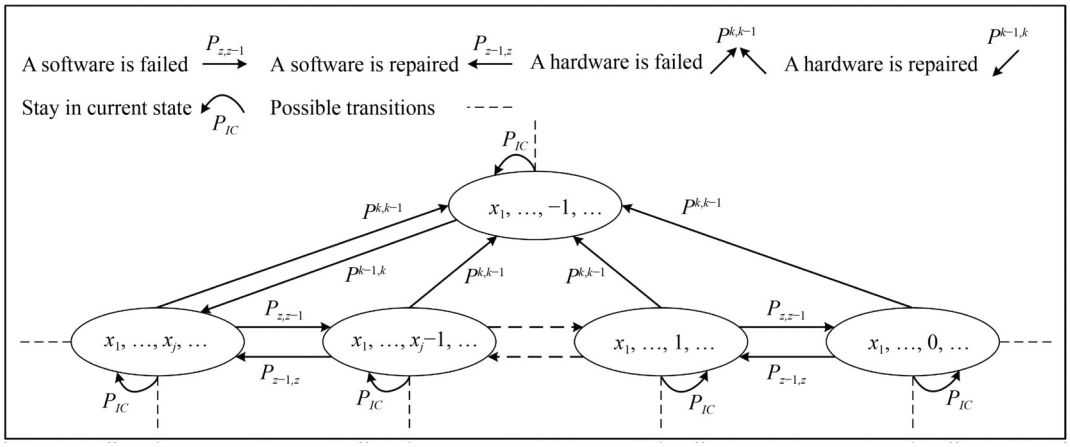

Fig. 2 Illustrations of cloud system state transitions.

confined to one of five possible conditions: 1) hardware failure transpires with a probability $P^{k,k-1}$; 2) hardware is repaired with a probability $P^{k-1,k}$; 3) software failure arises with a probability $P_{z,z-1}$; 4) software is repaired with a probability $P_{z,z+1}$; and 5) the system remains in the current state with a probability P_{IC}. The transient-state transfer probabilities during the Δt sojourn interval can be articulated as follows:

$$P^{k,\,k-1} = k\lambda_h \Delta t, \tag{4}$$

$$P^{k,\,k+1} = (m-k)\mu_h \Delta t, \tag{5}$$

$$P_{z,\,z-1} = z\lambda_s \Delta t, \tag{6}$$

$$P_{z,\,z+1} = \sum_i \sum_j (a_{ij} b_{ij}) \mu_s \Delta t, \tag{7}$$

$$P_{IC} = 1 - P^{k,\,k-1} - P^{k,\,k+1} - P_{z,\,z-1} - P_{z,\,z+1}, \tag{8}$$

where the number of active nodes k and running subservices z can be calculated as follows:

$$k = m - \sum_j h_j, \tag{9}$$

$$z = \sum_j [(1-h_j) \sum_i (1-b_{ij})]. \tag{10}$$

The probabilities associated with hardware and software failure/repair are depicted in Eqs. (4)–(7). Equation (8) enforces the constraint that only a single transition can transpire within the sojourn time interval Δt. It is important to observe that the defined probabilities predominantly comprise three constituent elements: The quantity of relevant hardware and software components (denoted as k and z), the failure and repair rates (represented as λ_h and μ_h), and the duration of the transition sojourn interval (Δt). Equations (9) and (10) were employed to compute the count of relevant components. It is worth noting that in scenarios where a service is interrupted due to a hardware failure, the only viable approach to recommence the service involves repairing the respective node (state transition from −1 to 1). Consequently, services impacted by hardware failures were not accounted for in Eqs. (6), (7), and (10).

3.4 System evaluations

In this context, we address the concepts of system reliability, performance, and PC prominent metrics often delineated in SLAs or pivotal to the operational costs of CSPs. System reliability pertains to the proportion of services successfully completed within their stipulated deadline requirements, a parameter intrinsically linked to user experience and customer contentment. For gauging system performance, we employed the aggregate completion time as a yardstick, which also provides insights into system throughput, with an emphasis on CSPs' perspectives. To assess system power consumption, we estimated the electric power consumed during service execution.

In contemporary computing architectures, the allocation of a service to multiple nodes can harness the advantages of parallel computing. The partitioning of a service into subservices is accomplished by distributing the service's workload requirements across distinct nodes. The workload requirement signifies the exertion needed for service completion and can be quantified as the product of resource volume and duration, akin to core hours. Jian et al. (2021) introduced a model to predict service duration based on users' historical data. For in-depth insights into service splitting, refer to the works of Zhang et al. (2020) and Bora et al. (2023). Each subservice on a separate physical node account for a portion of the total demand and operates autonomously (Xu et al., 2021). Presuming that workload requirements can be effectively divided, the subrequirements span from 0 to the total requirement. The execution time of each subservice hinges on its workload allocation and the quantum of allocated resources. Upon completion of a subservice, the associated physical node relinquishes the resources it occupied, with

the final subservice determining the overall service completion time (Qiu et al., 2021). This study encompasses the consideration of performance degradation owing to hardware resource contention among colocated services, as well as the performance enhancement stemming from service parallelism. We normalize the total speed of a physical node to unity, and the effective processing speed of each subservice is the outcome of Eqs. (2) and (3). For the estimation of PC, in line with prior investigations (Bennaceur and Kloul, 2020), we posit that a physical node's power conforms to a linear function of its present resource utilization. In Eq. (11), p_0 and p_1 are the static and peak power of the node, respectively.

$$Power = p_0 + (p_1 - p_0)\, utilization. \tag{11}$$

Furthermore, we have stipulated that in the event of a software failure, the system's power remains unaffected due to the continued operation of the backup image. Conversely, when a hardware failure arises, the system's power remains fixed at the static power level p_0, given that all backups are lost (Qiu et al., 2019). Subsequently, a node devoid of incomplete services is powered down, incurring zero PC.

Building upon the aforementioned definitions and presumptions, the crux of evaluating the system metrics rests on estimating time durations, encompassing parameters such as total service time and fault interruption time. Each physical node's resource capacity, denoted as c, constitutes a scalarized resource unit that amalgamates processor, memory, and hard disk attributes. A single resource unit serves as the smallest detachable unit within the virtualization process. The work demand and deadline requisites for service s_i are represented by w_i and d_i, respectively. Specifically, w_{ij} signifies workload apportioned to subservice s_{ij}, while the integer r_{ij} designates the count of scalarized resources allocated to it. It is pertinent to distinguish between workload and resource sharing. The former quantifies the exertion required for completing a service, generally quantified as core hours — the product of occupied processor cores and occupied hours. The latter centers on the number of resources, such as processor cores. We consider a one-time service assignment decision, wherein w_{ij} and r_{ij} remain constant until the culmination of s_{ij}. Let wp_{ijt} denote the service progress at time t, Eqs. (12) and (13) elucidate the resource releases:

$$wp_{ijt} = \begin{cases} 0 & \text{if } t = 0 \\ wp_{ijt-1} + r_{ijt-1} v_{ijt-1} & \text{otherwise} \end{cases}, \tag{12}$$

$$r_{ijt} = \begin{cases} r_{ijt-1} & \text{if } wp_{ijt} < w_{ij} \\ 0 & \text{otherwise} \end{cases}, \tag{13}$$

where v_{ijt} is the actual processing speed of subservice s_{ij} at time t, which is the product of the speed degradation of node n_j owing to resource contention and the speedup of service s_i by parallelism. Combining Eqs. (12) and (13), v_{ijt} is

$$\begin{aligned} v_{ijt} &= a_{ijt} v_{it} v_{jt} \\ &= \min\{r_{ijt}, 1\}(\textstyle\sum_j \max\{r_{ijt}, 1\})^{\rho}(\sum_i \max\{r_{ijt}, 1\})^{-\alpha}. \end{aligned} \tag{14}$$

Figure 1 illustrates the entire service execution process. During this process, the service may restart l times owing to random hardware failures or may be interrupted by random software failures. Thus, the total service time T_{ij}^s of s_{ij} consists of three parts: 1) the time interval T_{ij}^l when the most recent hardware failure is fixed and starts the final execution, 2) the time delay in the last execution due to software failures T_{ij}^d, and 3) the minimum time needed T_{ij}^w to complete the assigned workload, as shown in Eq. (15):

$$T_{ij}^s = T_{ij}^l + T_{ij}^d + T_{ij}^w, \tag{15}$$

where T_{ij}^w depends only on how the services are assigned, and T_{ij}^l and T_{ij}^d constitute uncertain random variables attributable to system availability. Given that availability correlates with system failure and repair rates, in conjunction with the duration of the service period, it holds true that irrespective of the number of services or servers involved, T_{ij}^l and T_{ij}^d can always be regarded as a proportion of T_{ij}^s and T_{ij}^w, respectively. As depicted in Eqs. (16) and (17), we utilize θ to denote the proportion of autoretry durations during service execution and η to denote the proportion of software downtimes in a hardware failure-free interval. Consequently, Eq. (18) emerges as an illustrative avenue for computing the total service time through assignment T_{ij}^w induced and availability-linked factors.

$$T_{ij}^l = \theta T_{ij}^s, \tag{16}$$

$$T_{ij}^d = \eta T_{ij}^w, \tag{17}$$

$$T_{ij}^s = \frac{1+\eta}{1-\theta} T_{ij}^w. \tag{18}$$

At any specific time t, the minimum service window of node n_j is established through Eq. (19). Within a number of $\lceil T_{jt}/\Delta t \rceil$ time intervals, we concentrate on the state transitions of each service between failures and repairs, excluding the influence of service completion on speed or other variables, as depicted in Fig. 2.

$$T_{jt} = \min\left\{ \frac{w_{ij} - wp_{ijt}}{r_{ijt} v_{ijt}},\ i = 1, 2, ..., |S_t| \right\}. \tag{19}$$

Figure 3 depicts the dynamic progression of service completion over a given time span. We assume the presence of three services on a node during the initial time slot.

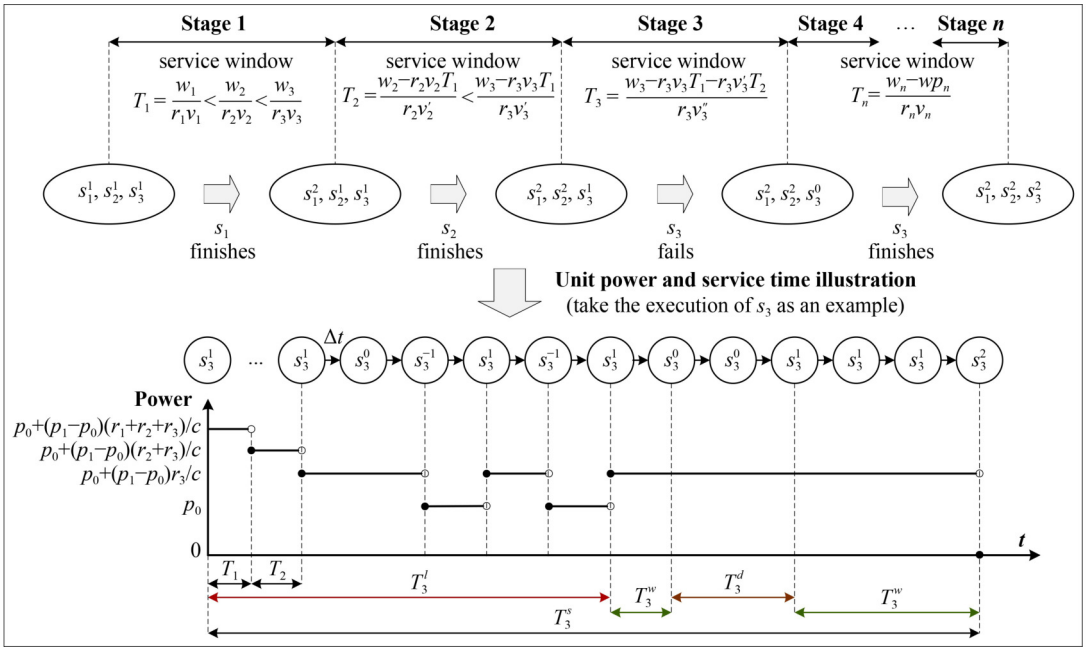

Fig. 3 Dynamic service completions and power changes.

The subscripts and superscripts denote the service index and service state, respectively. Without loss of generality, we assume that s_1 finishes first, then s_2 and finally s_3. Dynamic service completion impacts the system in two main ways. The first factor is the alteration in the number of services and active nodes within the system. According to Eqs. (4)–(10), such changes lead to fluctuations in transient probabilities. The second factor is the processing speed of the remaining services. Referring to Eq. (14), if any subservice is completed within a service window, the speed of the remaining subservices is adjusted. Consequently, a multistage system modeling approach is necessary to ensure stability within each service window. Equation (19) defines the service window, namely, T_1, T_2, and T_3 for stages 1 to 3.

Each service window indicates the minimum remaining operational time for unfinished services, representing the time needed to complete the remaining workload without any failure-induced downtime. Following each stage, at most one service achieves completion. For stages 1 and 2, we assume the absence of failures, and after stage 1, the resource r_1 allocated to s_1 is released. During the determination of the service window for stage 2, the speeds of s_2 and s_3 are adjusted in accordance with Eq. (14). Similarly, during stage 3, the speed of s_3 is recalibrated, resulting in its progress being divided into two segments with distinct speeds and durations. However, if a random failure occurs during stage 3, the completion of s_3 after stage 3 is impeded. Consequently, for potential stages 4 to n, it becomes necessary to redefine the progress of s_3 and the corresponding service window based on the potential effects of failures, encompassing failure types and durations.

Thus, T_3^s, the total service time of s_3, spans from the initiation of stage 1 to the conclusion of stage n. T_3^l denotes the time up to the epochs when the last hardware failure of s_3 is repaired. This value can be formulated as a proportion of T_3^s through Eq. (16), which relies on the distribution of system failures and repairs for both hardware and software. The downtime T_3^d caused by software failures, as depicted in Fig. 3, constitutes the proportion of workload requiring a time T_3^w in Eq. (17). T_3^s comprises T_3^l, T_3^d, and T_3^w, as expressed in Eqs. (15)–(18).

Moreover, node PC manifests in three scenarios: 1) shutdown with no PC, 2) solely static PC during hardware failures, and 3) PC according to a linear model with utilization. Analogous to the software downtime estimator η, we introduce β to denote the proportion of downtime attributable to hardware failures within a specified time range. β functions as an availability-related parameter for power estimation, determining whether the power remains static. θ and η serve as availability-related parameters for estimating service time.

Equation (20) for system reliability (RE) and Eq. (21) for system performance (PE) are calculated based on the ratio of on-time services and the summation of service times, respectively. Equation (22) for system power consumption is computed by multiplying the unit power and the active time for each node. The active time of node n_j is influenced by the most recently completed service, denoted $\max_j\{T_{ij}^s\}$. The indicator function $I(\cdot)$ in Eq. (20) assumes the value of 1 when the proposition $(\cdot)$ is true and 0 otherwise. The unit power for each server at time t is derived using Eq. (11), factoring in utilization calculations and the effects of hardware failure downtime.

$$RE = \frac{\sum_i I(\max_j\{T_{ij}^s\} \leqslant d_i)}{|S_0|}, \tag{20}$$

$$PE = \sum_i \max_j\{T_{ij}^s\}, \tag{21}$$

$$PC = \sum_j \sum_{t=0}^{\max_j\{T_{ij}^s\}} \left[p_0 + (1-\beta)(p_1 - p_0)\left(\sum_i r_{ijt}\right)/c\right]. \tag{22}$$

4 Estimation method

In this section, we delve into the methodology for estimating the probability density distribution of the pertinent metrics and availability-related factors using a multistage SPR approach.

The SRP algorithm was introduced by Du et al. (2015) for estimating the transient downtime distribution of VMs with allocated backups to ensure service availability. This algorithm generates an adequate number of sample paths randomly by employing a predefined state-transfer probability matrix. The overall distribution is then estimated using the observations generated in the samples, which are sufficiently numerous to meet convergence criteria. These estimators possess favorable statistical properties such as unbiasedness, consistency, and sufficiency. In a similar vein, Guo et al. (2019b) proposed a dynamic backup strategy based on SPR for downtime distribution estimation, extending the SPR algorithm framework to encompass the estimation of downtime distributions for scenarios involving multiple virtualized network functions (Guo et al., 2020). In alignment with these works, our study proposes the expansion of the SPR algorithm into a multistage design to accommodate the dynamic service completions based on the transient probabilities delineated in Eqs. (4)–(10), as well as the service window identification presented in Eq. (19).

Algorithm 1 illustrates the state transitions in path generation. The system state is represented in a matrix form $\boldsymbol{M}$, where each row denotes an unfinished service, and each column signifies a physical node. The corresponding entry M_{ij} conveys the state of subservice s_{ij} (with values of −1, 0, or 1). Initially, we calculate the number of uptime and downtime facilities using Eqs. (9) and (10). Subsequently, leveraging predefined failure and repair rates, as well as the sojourn time interval, we derive the transient-state transfer probabilities through Eqs. (4)–(8). A random number is generated from a uniform distribution spanning 0 to 1, and its comparison with the cumulative probability of the transition probabilities determines the occurrence of a state transition. Finally, the system state undergoes updating in line with the birth–death process outlined in Subsection 3.3, leading to the creation of a new state matrix $\boldsymbol{M}$.

Algorithm 2 outlines the multistage SPR methodology. First, Eq. (19) is employed to generate individual sample paths. Following this, each sample path updates its states at each time interval by following the logic laid out in

Algorithm 1 State transition in path generation

Input: Current state matrix $\boldsymbol{M}$, sojourn time interval Δt

Output: The next state matrix $\boldsymbol{M}$

1. Get the uptime nodes set N and hardware failed nodes set $\overline{N}$ //using Eq. (9)
2. Get the uptime services set S_t and software failed services set $\overline{S}_t$ //using Eq. (10)
3. Update transient probability matrix $\boldsymbol{P}$ by the size of N, $\overline{N}$, S_t, and $\overline{S}_t$ //using Eqs. (4)–(8)
4. Randomly generate a variable δ from the *Uniform*(0, 1)
5. **If** $\delta \leqslant P^{k,k-1}$ // a hardware failure occurs
6. Randomly select a server j from N
7. **For** service i in j
8. $M_{ij} \leftarrow -1$
9. **Else if** $\delta \leqslant P^{k,k-1} + P^{k,k+1}$ // a hardware failure if repaired
10. Randomly select a server j from $\overline{N}$
11. **For** service i in j
12. $M_{ij} \leftarrow 1$
13. **Else if** $\delta \leqslant P^{k,k-1} + P^{k,k+1} + P_{z,z-1}$ // a software failure occurs
14. Randomly select a service ij from S_t
15. $M_{ij} \leftarrow 0$
16. **Else if** $\delta \leqslant P^{k,k-1} + P^{k,k+1} + P_{z,z-1} + P_{z,z+1}$ // a software failure is repaired
17. Randomly select a service ij from $\overline{S}_t$
18. $M_{ij} \leftarrow 1$
19. **Else** // stay in current states
20. **Pass**

Return $\boldsymbol{M}$

Algorithm 2 Multistage SPR algorithm

Input: Resource assignment matrix $\boldsymbol{A}$, workload assignment matrix $\boldsymbol{W}$, scalability factor ρ, coherency factor α, time limit T, time interval Δt, initial sample size L, and convergence criterion ε

Output: The sample mean and standard deviation of probability densities of concerned metric τ

1. **For** $l = 1$ to L
2. Initialize the states matrix $\boldsymbol{M} \leftarrow \boldsymbol{A}$, and time $t \leftarrow 0$
3. **For** node $j = 1$ to N
4. **While** $t < T$
5. Calculate current service window $SW \leftarrow \min\{T - t, T_{jt}\}$ //using Eqs. (14)–(19)
6. **Call** Algorithm 1 to generate a path with $\lceil SW/\Delta t \rceil$ iterations
7. $t \leftarrow t + SW$, and update the service process with $\boldsymbol{W}$ //using Eqs. (12)–(14)
8. Calculate the sample means $\hat{u}_L(\tau') = \sum_{l=1}^{L} I(\tau = \tau') \Big/ L$ and sample variances $\hat{\sigma}_L^2(\tau') = \left[\sum_{l=1}^{L} I(\tau = \tau') - \hat{u}_L(\tau')\right] \Big/ (L-1)$
9. **Repeat**
10. Generate one more random sample path
11. Calculate the sample means $\hat{u}_{L+1}(\tau')$ and sample variances $\hat{\sigma}_{L+1}^2(\tau')$
12. **If** $\max\left\{\left|\hat{u}_{L+1}(\tau') - \hat{u}_L(\tau')\right| \Big/ \hat{u}_L(\tau'), \left|\hat{\sigma}_{L+1}^2(\tau') - \hat{\sigma}_L^2(\tau')\right| \Big/ \hat{\sigma}_L^2(\tau')\right\} > \varepsilon$
13. $L \leftarrow L + 1$, $\hat{u}_L(\tau') \leftarrow \hat{u}_{L+1}(\tau')$, $\hat{\sigma}_L^2(\tau') \leftarrow \hat{\sigma}_{L+1}^2(\tau')$
14. **Else**
15. **Break**

Return $\hat{u}_L(\tau')$, $\hat{\sigma}_L^2(\tau')$

Algorithm 1. Upon the culmination of each stage, we update the current service processes and proceed to the subsequent stage until the predefined time limit is attained. Upon generating all sample paths, we calculate the sample mean and variance for each feasible value. These estimated values are deemed acceptable if the alterations in sample means and variances from the freshly generated samples are adequately small. Conversely, if the changes are substantial, new samples are continuously generated until a convergence criterion is met.

5 Numerical experiments

This section outlines a series of numerical experiments undertaken to exemplify the utilization of the introduced framework. Initially, we juxtapose the resultant system metrics arising from diverse service assignment policies that integrate resource sharing and service parallelism. Subsequently, we formulate optimization models for service assignment to strike a balance between performance, PC, and reliability assurance. Finally, we contrast the outcomes of optimal assignments by incorporating various availability-related factors and workload intensities.

5.1 Comparison of different assignments

In the experimental setup, we consider four homogenous services, each requiring four core hours and having a deadline within one hour. The system comprises eight homogenous servers, each equipped with eight processing cores. Random failures and repairs are assumed to follow exponential distributions with parameters $\lambda_h = 0.005$, $\lambda_s = 0.04$, $\mu_h = 2.5$, and $\mu_s = 0.5$ (h^{-1}), following Qiu et al. (2019). The power values of $p_0 = 133.2$ and $p_1 = 222$ (W) are taken from Al-Moalmi et al. (2021). For simplicity, in this example, the assigned resource volume is directly proportional to the workload, implying that one processing core corresponds to one core-hour workload. However, this constraint can be relaxed in subsequent optimization analyses. Four distinct service assignment policies, each considering resource sharing and service parallelism, are devised: Policy 1 with neither parallelized service nor resource sharing, policy 2 with resource sharing only, policy 3 with service parallelism only, and policy 4 with both parallelized service and resource sharing. Appendix B in the Supplementary Material provides further detailed information. Following the settings in Guo et al. (2020) for transient downtime estimation, we set the sojourn time interval, convergence criterion, and initial sample size to $\Delta t = 0.005$ (h), $\varepsilon = 0.001$, and $L = 20000$, respectively. The scalability factor ρ and coherency factor α are both set to 0.2.

Table 1 illustrates the results of the numerical experiments. Using policy 1 outcomes as a baseline, we observe that policies 2, 3, and 4 lead to reduced reliability. Policy 2, due to its lower processing speed, prevents services from meeting their deadlines. In terms of performance, policies 3 and 4 significantly reduce the total service time by almost 12% compared to policy 1, while policy 2

increases the time by approximately 15%. However, concerning PC, policy 2 saves nearly 30% more energy than policy 1, while policies 3 and 4 consume more energy. This can be explained by the following factors. Resource sharing consolidates services onto limited nodes, making them susceptible to hardware failures and subsequent state transitions. Meanwhile, service parallelism distributes the workload across multiple nodes, which increases the risk of failures with more devices involved. Additionally, the service time itself affects reliability. Higher resource contention results in slower processing speeds, whereas service parallelism accelerates execution, potentially leading to an increased risk of failure due to the longer duration.

Although the numerical experiments were conducted under ideal conditions, they offer valuable insights. Parallelism proves beneficial in reducing service time, a crucial factor when availability levels are relatively low. The time saved can offset fault-related downtimes, thereby ensuring system reliability and performance. Resource sharing, while facing performance losses and reliability risks, offers significant energy savings. Additionally, when the workload approaches the resource's capacity, resource sharing becomes essential. PC is the outcome of multiplying the unit power and service time. Service parallelism reduces service time but increases unit power, whereas resource sharing has the opposite effect. The synergy of service parallelism and resource sharing can potentially counterbalance these effects. As depicted in Fig. 4, a higher probability of failure can decrease system availability, thereby jeopardizing system reliability by prolonging service time. CSP administrators can mitigate the risk of reliability breaches by employing service parallelism and resource sharing strategies. Although parallelism boosts PC, it reduces time, leading to enhanced performance. On the other hand, resource sharing lowers PC (due to the substantial contribution of a node's static power) but can hinder performance. Thus, the central trade-off between performance and PC mainly revolves around unit power and service time, both of which are significantly influenced by the design of service parallelism and resource sharing mechanisms.

5.2 Optimal service assignments

We employ optimization models to compare optimal decisions across various contexts and capture interrelations among metrics. First, we define the demand and capacity constraints as presented in Eqs. (23) and (24), respectively. The value range of r_{ijt} is outlined in Eq. (25), which must be a nonnegative integer not exceeding the capacity limitation. It is important to note that all these constraints are established in correspondence with $i \in \{0, 1, ..., |S_t|\}$, $j \in \{0, 1, ..., |N|\}$, and $t \in \{0, 1, ..., \lceil T/\Delta t \rceil\}$.

$$\sum_j w_{ij} = w_i, \tag{23}$$

$$\sum_i r_{ijt} \leqslant c, \tag{24}$$

Table 1 Outcomes of different service assignment policies

Policies	Reliability (%)		Performance (h)		Power consumption (kWh)	
	Best	Expectation	Best	Expectation	Best	Expectation
Policy 1	100	**99.500**	4.000	4.010	0.710	0.711
Policy 2	0	0	4.600	4.608	0.510	**0.511**
Policy 3	100	99.090	3.500	**3.523**	1.088	1.091
Policy 4	100	98.140	3.500	3.545	1.010	1.013

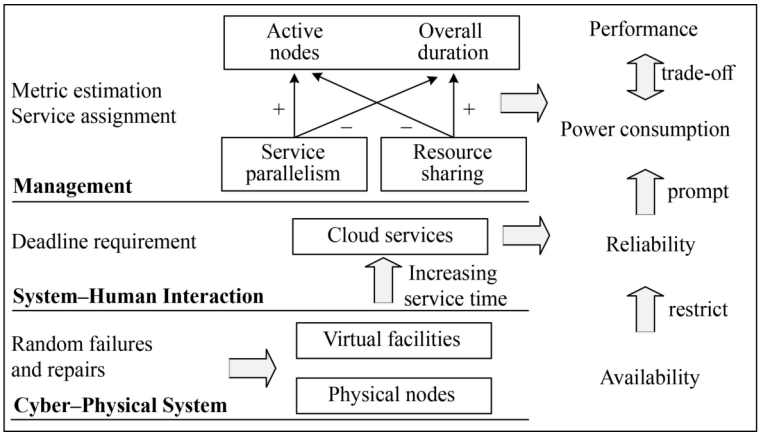

Fig. 4 Correlations between system metrics.

$$r_{ijt} \in \{0, 1, 2, ..., c\}. \quad (25)$$

In terms of the system metrics, we accord the utmost priority to system reliability. The ability to deliver services according to the schedule is directly linked to the reputation of the CSP, a critical factor in a fiercely competitive cloud service market (Guan et al., 2020). Consequently, we establish a reliability constraint as presented in Eq. (26).

$$\max_j\{T_{ij}^s\} \leqslant d_i. \quad (26)$$

The service assignment can then be optimized by minimizing the objective functions for *PE* and *PC* from Eqs. (21) and (22), respectively.

(**Performance Opt.**)

Minimize Eq. (21)

(**Power Opt.**)

Minimize Eq. (22)

Both of these metrics are constrained by $RE = 1$, Eqs. (12)–(18), and Eqs. (23)–(26).

The availability-related factors, namely, 1) the proportion of autoretries (θ), 2) the proportion of software failure-caused downtimes (η), and 3) the proportion of hardware failure-caused downtimes (β), are estimated through the multistage SPR approach using a time length of one hour. Notably, these availability-related factors are solely determined by the distribution of random failures and repairs. Given that random failures and repairs are assumed to be independent, the values of the availability-related factors remain unaffected by the number of physical nodes, services, and assignment decisions. Figure 5 illustrates the comparisons of the distributions for different failure rates. Panels (a1), (b1), and (c1) display the estimated $\hat{\theta}$, $\hat{\eta}$, and $\hat{\beta}$ values from various software failure rates, respectively. Panels (a2), (b2), and (c2) present the comparisons among different hardware failure rates. In all panels, the probability density functions of the corresponding estimation distributions are depicted; a steeper curve indicates a more concentrated estimated factor distribution, and vice versa.

In panels (a1) and (c1) of Fig. 5, the hardware-related factors θ and β remain relatively stable across varying software availability levels. The software-related factor η shown in panel (b1) is closely linked to software failures. Although $\hat{\eta}$ under different software availability conditions is mostly concentrated within 0.01, as the software failure rate increases, the distribution of $\hat{\eta}$ becomes broader. However, the relative change in its distribution is not linear with an increase in the failure rate. With each increment of 0.02 in the software failure rate, the range of $\hat{\eta}$ gradually decreases.

Panels (a2) and (c2) also demonstrate outcomes similar to panel (b1) under different hardware failure conditions. Since η is defined as the proportion of software failure downtime within the final retry process, it is estimated from a period without hardware failures. Consequently, the distribution of $\hat{\eta}$ also shifts with varying hardware failure rates in panel (b2). The results in Fig. 5 are based on exponentially distributed random failures and repairs. The failure rates are considered to be time-independent due to the relatively short service times. Furthermore, we also showcase the estimations under other nonconstant failure rates, such as the Weibull and Erlang distribution

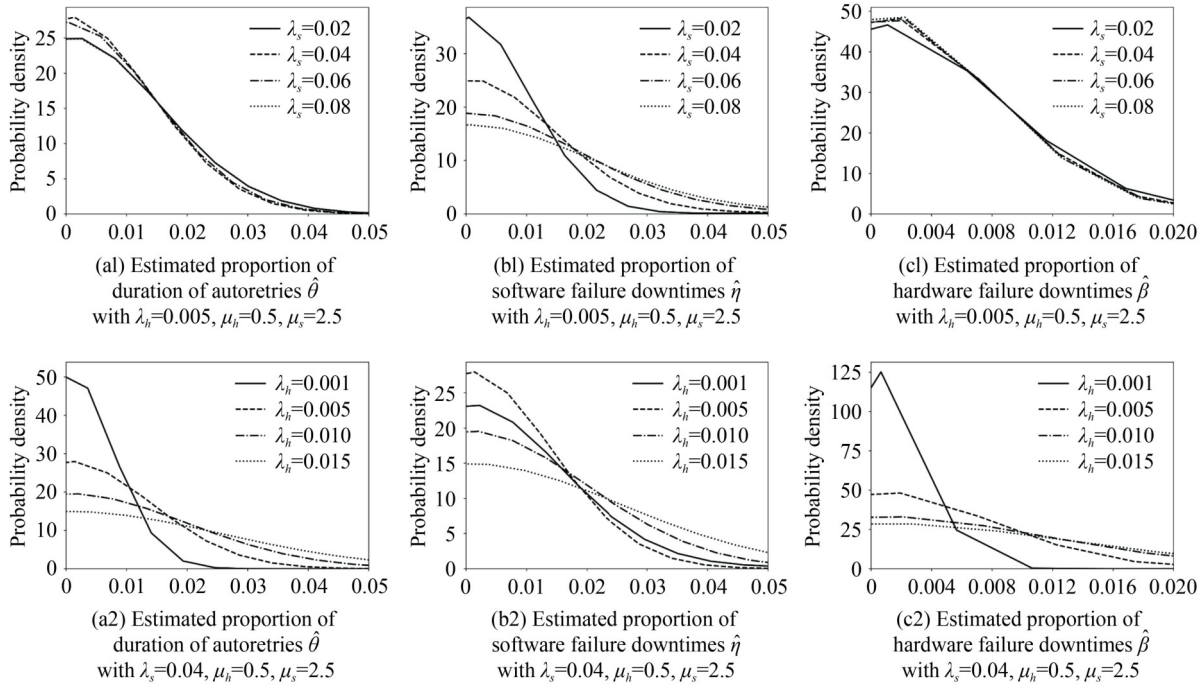

Fig. 5 Estimated distribution for availability factors.

models, in Appendix C in the Supplementary Material. These models can capture hardware degradation phenomena as well (Guo et al., 2020).

The focus here was not on controlling availability but rather on detecting the effects of availability conditions on other system metrics and analyzing the trade-offs between performance and PC. Therefore, in the following optimization experiments, we applied three combinations of availability-related factors: 1) $\hat{\theta}_1 = 0$, $\hat{\eta}_1 = 0$, and $\hat{\beta}_1 = 0$; 2) $\hat{\theta}_2 = 0.004$, $\hat{\eta}_2 = 0.01$, and $\hat{\beta}_2 = 0.002$; and 3) $\hat{\theta}_3 = 0.005$, $\hat{\eta}_3 = 0.015$, and $\hat{\beta}_3 = 0.004$. These different combinations of availability-related factors can be seen as representing systems with varying failure and repair rates or CSP administrators with heterogeneous risk preferences for service assignments. Higher values indicate more downtimes during service execution, indicating more stringent availability conditions or more conservative decisions.

Similar to the example in Section 5.1, we assumed eight homogenous physical nodes, each with eight processor cores. To analyze decision-making under different workload levels, we considered three workload traces: 1) two services with six core-hour requirements each, representing a light workload condition with 25% capacity; 2) four services with six core-hour requirements each, representing a moderate workload condition with 50% capacity; and 3) six services with six core-hour requirements each, indicating an intensive workload with 75% capacity. The deadline requirements for all services were set within one hour, and we used a time window of $T = 1.05$ and a time interval of $\Delta t = 0.05$ (h). The optimization models were solved using the Gurobi optimizer.

Figures 6 and 7 provide a comparison of the optimal solutions from different objective models under varying workload levels and availability conditions. Figure 6 illustrates the optimal solutions obtained from different objectives and calculates their relative gains and losses using Eqs. (27) and (28). The subscript indicates the value obtained from the respective objective solution.

$$TimeSaving = \frac{PE_{\text{power}} - PE_{\text{performance}}}{PE_{\text{performance}}}, \tag{27}$$

$$EnergySaving = \frac{PC_{\text{performance}} - PC_{\text{power}}}{PC_{\text{power}}}. \tag{28}$$

As the workload increases, the potential time savings decrease, while the energy savings become more pronounced. When the workload is relatively low, service parallelism significantly reduces the service time with lower energy consumption. However, when the workload becomes sufficiently high, the system's resource capacity limits the number of parallel services that can be effectively utilized. Observing Fig. 6, when the workload is only 25% of the system's capacity, over half of the time can be saved with an approximately 10% increase in energy consumption. However, as the workload increases to 75%, the potential performance improvement decreases steeply to a range of 10%–30%, while the potential energy savings increase to more than 20%.

Moreover, the solid, dashed, and dash-dotted lines in Fig. 6 represent different scenarios of system availability, ranging from high to low. Regardless of the availability conditions, the trends in optimal decision-making with respect to changes in workload remain consistent. As availability decreases, the potential time and energy savings are diminished. This observation illustrates the impact of availability constraints on system performance and PC.

Figure 7 further compares the magnitude of parallelism (Eq. (29)) and sharing (Eq. (30)) under different decisions.

$$Parallelism = \frac{\sum_i \sum_j a_{ij}}{|S|}, \tag{29}$$

$$Sharing = \frac{\sum_i \sum_j a_{ij}}{m}. \tag{30}$$

As depicted in Fig. 7, the extent of service parallelism

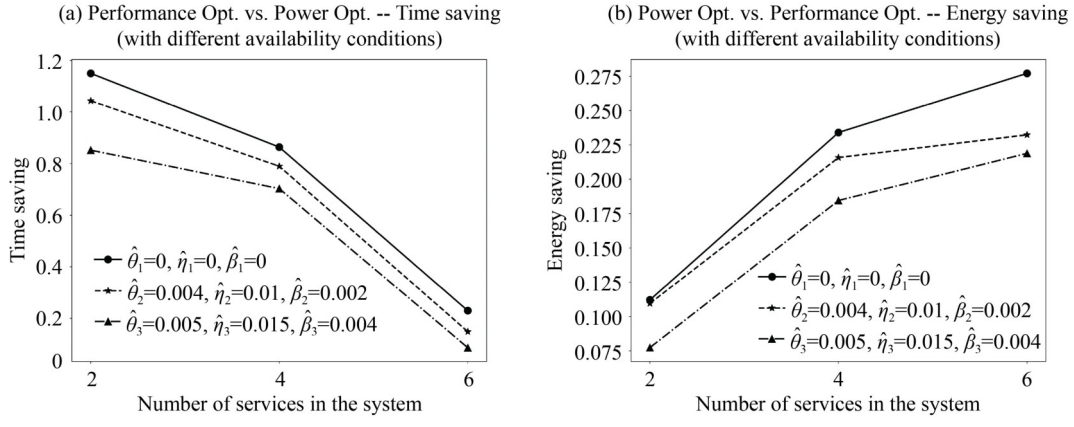

Fig. 6 Performance-power consumption trade-offs (with $\alpha = \rho = 0.2$).

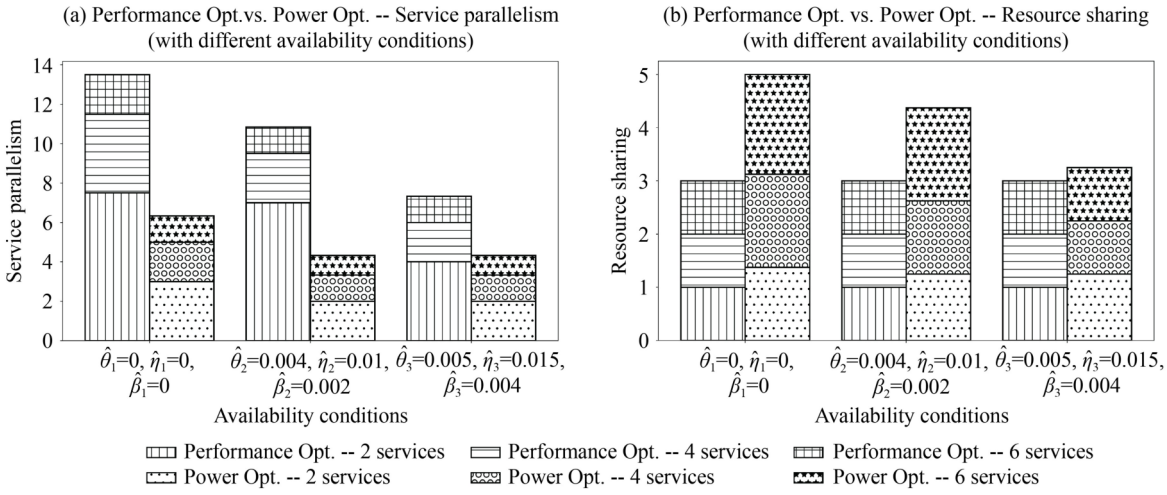

Fig. 7 Resource sharing and service parallelism designs (with $\alpha = \rho = 0.2$).

and resource sharing diminishes gradually as workload intensity increases and availability deteriorates. The former scenario highlights resource limitations. In the context of performance optimization, when the workload is relatively light, services tend to be executed in parallel. However, as the workload approaches the system's capacity, constrained resources make it challenging to maintain high levels of parallelism. In contrast, when focusing on performance optimization, it becomes prudent to avoid excessive resource sharing to minimize speed loss. PC optimization, on the other hand, carefully employs parallel services and selects resource sharing to mitigate unit PC. The latter scenario addresses the constraints imposed by limited availability. This explanation also aligns with the potential time and energy savings demonstrated in Fig. 6, where a decrease in availability leads to diminished opportunities for optimizing time and energy consumption.

Figures 6 and 7 illustrate the outcomes with a coherency factor α and scalability factor ρ set to 0.2. Similarly, akin to the analysis of the impact of availability conditions on service assignment, Figs. 8 and 9 compare the influences of various service characteristics. It is important to recall that the coherency factor α characterizes the resource utilization patterns of colocated services, with a larger α leading to increased resource contention and performance degradation. Meanwhile, the scalability factor ρ signifies the extent to which a service can benefit from parallelism, depending on its architectural design.

Similar to Fig. 6, Fig. 8 demonstrates a pattern of diminishing performance gains and expanding energy saving potential as workload increases. Consequently, the effects of service characteristics on performance and power optimization models are diametrically opposed. Examining the impact of the scalability factor (panels (a1) and (a2)), services more amenable to parallel operation stand to gain more from performance optimization. Nonetheless, the effects taper off as the workload surpasses half of the system's capacity due to resource limitations. A higher scalability factor also compresses the space for energy savings. On the other hand, concerning the coherency factor (panels (b1) and (b2)), if services tend to exhibit more homogeneous resource demand and usage, the PC model reaps benefits while the performance model experiences constraints.

Figure 9 furthers our understanding of how service characteristics influence decision-making. In comparison to panels (a1) and (b1), aggressive service parallelism becomes more prevalent as the scalability of services increases. However, this parallel design diminishes as coherency rises. Aggressive parallelism, owing to the constraints on physical nodes, might occupy all available nodes, prompting subsequent resource sharing. Contrasting panels (a2) and (b2), performance optimization decisions consistently steer clear of resource sharing, whereas power optimization shows a propensity toward moderate resource sharing.

Based on the optimization results, under conditions of low workload and favorable availability, a marginal increase in PC could yield significant performance enhancements. However, with mounting workloads or diminishing availability, the benefits of power optimization could intensify. The juxtaposition of performance and power is especially pronounced in scenarios of moderate usage and availability. CSP administrators must make decisions based on the specific costs of performance and PC. They might choose to mitigate resource sharing to enhance performance while optimizing parallelism, and vice versa, to curtail PC.

In summary, we compared the effects of availability conditions, workload intensity, and service characteristics on performance, power optimization decisions, and potential benefits. The key takeaways are as follows: 1) enhanced system performance primarily hinges on service parallelism and the avoidance of resource sharing; 2) reducing PC necessitates well-calibrated service

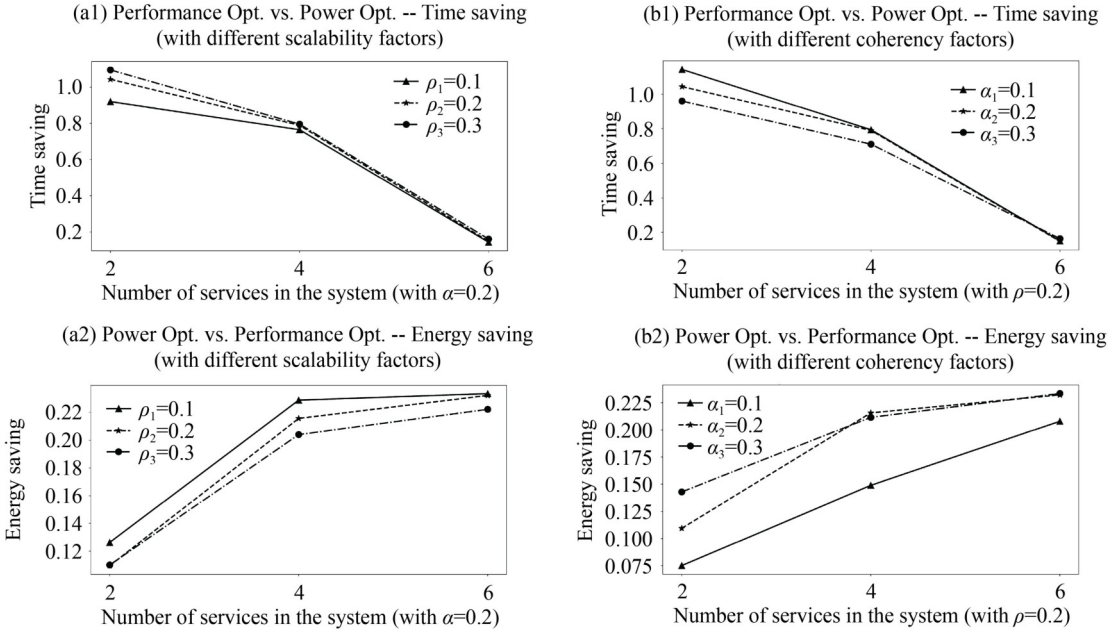

Fig. 8 Performance-power consumption trade-offs (with $\hat{\theta}_2 = 0.004$, $\hat{\eta}_2 = 0.01$, $\hat{\beta}_2 = 0.002$).

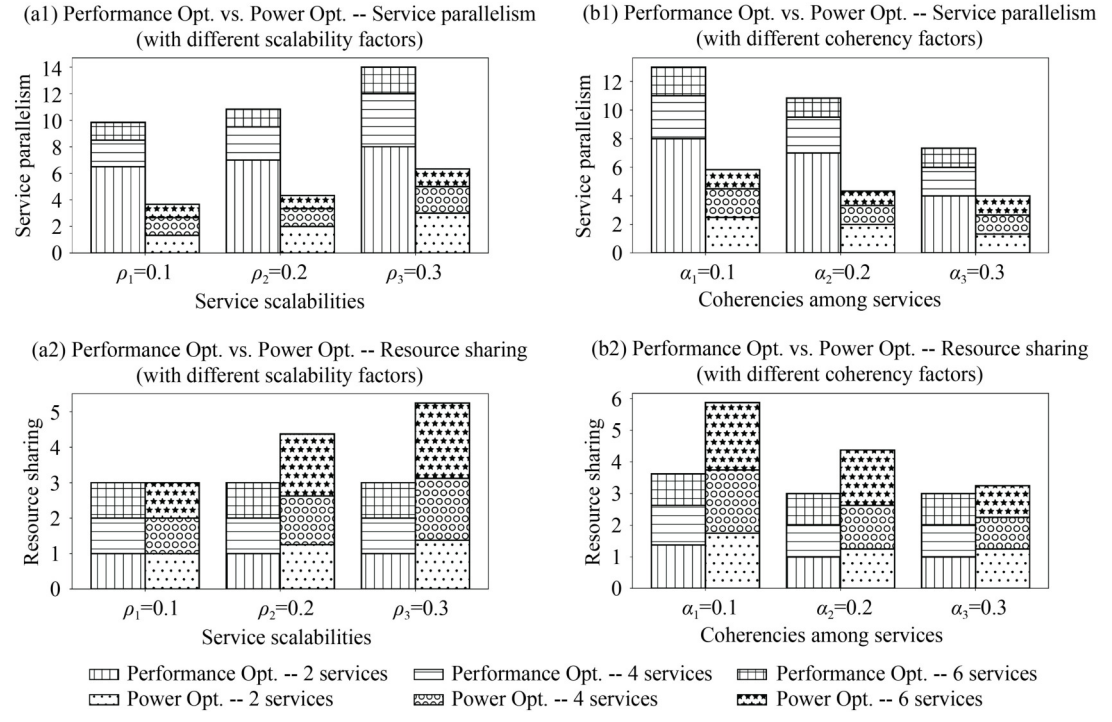

Fig. 9 Resource sharing and service parallelism designs (with $\hat{\theta}_2 = 0.004$, $\hat{\eta}_2 = 0.01$, $\hat{\beta}_2 = 0.002$).

parallelism and resource sharing designs, with the latter playing a more pronounced role; 3) diminished availability adversely affects both performance and power optimizations; 4) workload intensity significantly impacts potential benefits, with high workloads constraining performance optimization but amplifying energy-saving potential; and 5) the impact of service characteristics on performance and power optimization might be antithetical, implying that changes in service characteristics could lead to altered trade-offs between performance and PC.

6 Conclusions

Given the widespread adoption of virtualization technology, the cloud service environment has taken on the

characteristics of a CPS. CSP administrators face the task of allocating services while considering various metrics, including system reliability, performance, and PC. This study employs birth–death processes to model random software and hardware failures in a cloud-integrated CPS. In response to the challenges posed by virtualization, we delve into the influence of service parallelism and resource sharing on service processing speed and duration. The model indicates that while maintaining system reliability, the trade-off between system performance and PC centers around the interplay between unit power and service duration. Service parallelism can reduce service duration and enhance system performance but might elevate PC. Conversely, resource sharing can augment resource utilization and lower unit power, yet it decelerates the service rate. Leveraging an optimization model encompassing performance and PC, we conduct a comparative analysis of the effects of availability conditions, workload intensity, and service characteristics on decision-making. Numerical experiments reveal that availability can curtail the utilization of parallel services and resource sharing, thus constraining the outcomes of both performance and power optimization models. Workload intensity significantly influences performance optimization. As the workload increases, the potential performance gain decreases rapidly, while the energy-saving benefits gradually increase. For scenarios where service scalability is high, performance optimization proves more effective, although the advantage of power optimization diminishes. However, the impact of service coherence yields the opposite result.

This study presents a systematic framework for the analysis, evaluation, and optimization of system reliability, performance, and PC. The proposed framework establishes a connection between distinct system metrics and service duration, which, in turn, hinges on system availability and service allocation. System availability is embodied in the distribution of software and hardware failures and repairs, reflecting the repercussions of failure interruptions on service duration. We differentiate the effects of random hardware and software failures on service interruptions and employ transient state analysis, which is particularly suited for relatively brief service windows. The multistage SPR approach offers an efficient means to estimate service time with dynamic service completions. This framework equips CSP administrators and researchers to assess other pertinent execution attributes within cloud-integrated CPS.

The allocation of services is intricately tied to the parallelism and resource sharing design. We have taken into account the impact of service parallelism and resource sharing on the service processing rate. Through an array of numerical experiments, we scrutinize the influence of different availability conditions, workload intensities, and service characteristics on the trade-offs between performance and PC. This study holds managerial implications for cloud service assignments. In scenarios characterized by high availability and relatively low workload, optimizing system performance is crucial. However, potential performance gains considerably diminish as workload escalates or availability wanes. On the energy-saving front, while potential gains wane with availability, they escalate with workload levels. Resource sharing can curtail unit power while elongating service time, whereas service parallelism engenders the opposite outcome. Achieving a balance between these factors in service assignment decisions calls for meticulous design.

Several promising avenues are worthy of future research. First, this study primarily focuses on the allocation and dynamic completion of existing services. Subsequent investigations might amalgamate queuing optimization techniques to accommodate dynamic arrivals. Second, we assume that services are backed up in real time and seamlessly reinitiated after the repair of software failures. Alternative system availability assurance strategies, such as VM backups with delayed recovery, warrant analysis. Moreover, real-world deadline requirements might not be rigid constraints. Thus, accounting for aspects such as reliability penalties, overtime losses, and PC in a comprehensive manner could lead to the design of objective functions that simultaneously balance reliability, performance, and PC. Last, cloud service assignment is intrinsically an NP-hard problem, while cloud service systems boast complex, high-dimensional features, including dynamics, multitenancy, and multiresource specifications. Equally important is the development of efficient algorithms to address cloud service allocation. With the advent of artificial intelligence, future investigations can harness deep learning models to unravel the intricate relationships among cloud system availability, reliability, performance, and power.

Electronic Supplementary Material Supplementary material is available in the online version of this article at https://doi.org/10.1007/s42524-023-0272-2 and is accessible for authorized users.

Competing Interests The authors declare that they have no competing interests.

References

Al-Moalmi A, Luo J, Salah A, Li K, Yin L (2021). A whale optimization system for energy-efficient container placement in data centers. Expert Systems with Applications, 164: 113719

Amdahl G M (1967). Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the Spring Joint Computer Conference. Atlantic City, NJ: Association for Computing Machinery, 483–485

Ataie E, Entezari-Maleki R, Etesami S E, Egger B, Sousa L, Movaghar A (2022). Modeling and evaluation of dispatching policies in IaaS cloud data centers using SANs. Sustainable Computing: Informatics and Systems, 33: 100617

Bai X, Li M, Chen B, Tsai W T, Gao J (2011). Cloud testing tools. In: Proceedings of 6th International Symposium on Service Oriented System. Irvine, CA: IEEE, 1–12

Bennaceur W M, Kloul L (2020). Formal models for safety and performance analysis of a data center system. Reliability Engineering & System Safety, 193: 106643

Bora S, Walker B, Fidler M (2023). The tiny-tasks granularity trade-off: Balancing overhead versus performance in parallel systems. IEEE Transactions on Parallel and Distributed Systems, 34(4): 1128–1144

Canosa-Reyes R M, Tchernykh A, Cortés-Mendoza J M, Pulido-Gaytan B, Rivera-Rodriguez R, Lozano-Rizk J E, Concepcion-Morales E R, Castro Barrera H E, Barrios-Hernandez C J, Medrano-Jaimes F, Avetisyan A, Babenko M, Drozdov A Y (2022). Dynamic performance: Energy tradeoff consolidation with contention-aware resource provisioning in containerized clouds. PLoS One, 17(1): e0261856

Cao X, Bo H, Liu Y, Liu X (2023). Effects of different resource-sharing strategies in cloud manufacturing: A Stackelberg game-based approach. International Journal of Production Research, 61(2): 520–540

Chinnathambi S, Santhanam A, Rajarathinam J, Senthilkumar M (2019). Scheduling and checkpointing optimization algorithm for Byzantine fault tolerance in cloud clusters. Cluster Computing, 22(S6): 14637–14650

Cotroneo D, de Simone L, Liguori P, Natella R (2022). Fault injection analytics: A novel approach to discover failure modes in cloud-computing systems. IEEE Transactions on Dependable and Secure Computing, 19(3): 1476–1491

Du A Y, Smith S D, Yang Z, Qiao C, Ramesh R (2015). Predicting transient downtime in virtual server systems: An efficient sample path randomization approach. IEEE Transactions on Computers, 64(12): 3541–3554

Eshraghi N, Liang B (2019). Joint offloading decision and resource allocation with uncertain task computing requirement. In: IEEE Conference on Computer Communications. Paris: IEEE, 1414–1422

Fahmideh M, Beydoun G, Low G (2019). Experiential probabilistic assessment of cloud services. Information Sciences, 502: 510–524

Feng W, Huang M (2015). The research on service composition trust based on cloud computing. In: International Conference on Computer Science and Intelligent Communication. Zhengzhou: Atlantis Press, 291–294

Garg R, Mittal M, Son L H (2019). Reliability and energy efficient workflow scheduling in cloud environment. Cluster Computing, 22(4): 1283–1297

Guan Z, Ye T, Yin R (2020). Channel coordination under Nash bargaining fairness concerns in differential games of goodwill accumulation. European Journal of Operational Research, 285(3): 916–930

Guo J, Chang Z, Wang S, Ding H, Feng Y, Mao L, Bao Y (2019a). Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces. In: Proceedings of the 27th International Symposium on Quality of Service. Phoenix, AZ: IEEE, 1–10

Guo M, Guan Q, Chen W, Ji F, Peng Z (2022). Delay-optimal scheduling of VMs in a queueing cloud computing system with heterogeneous workloads. IEEE Transactions on Services Computing, 15(1): 110–123

Guo Z, Li J, Ramesh R (2019b). Optimal management of virtual infrastructures under flexible cloud service agreements. Information Systems Research, 30(4): 1424–1446

Guo Z, Li J, Ramesh R (2020). Scalable, adaptable, and fast estimation of transient downtime in virtual infrastructures using convex decomposition and sample path randomization. INFORMS Journal on Computing, 32(2): 321–345

Guo Z, Li J, Ramesh R (2023a). Green data analytics of supercomputing from massive sensor networks: Does workload distribution matter? Information Systems Research, 34(4): 1664–1685

Guo Z, Zhang Y, Liu S, Wang X V, Wang L (2023b). Exploring self-organization and self-adaption for smart manufacturing complex networks. Frontiers of Engineering Management, 10(2): 206–222

Gupta A, Acun B, Sarood O, Kalé L V (2014). Towards realizing the potential of malleable jobs. In: 21st International Conference on High Performance Computing. Goa: IEEE, 1–10

Han X, Schooley R, Mackenzie D, David O, Lloyd W J (2020). Characterizing public cloud resource contention to support virtual machine co-residency prediction. In: IEEE International Conference on Cloud Engineering. Sydney: IEEE, 162–172

Harchol-Balter M (2021). Open problems in queueing theory inspired by datacenter computing. Queueing Systems, 97(1–2): 3–37

Ibrahim M, Nabi S, Hussain R, Raza M S, Imran M, Kazmi S M A, Oracevic A, Hussain F (2020). A comparative analysis of task scheduling approaches in cloud computing. In: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing. Melbourne: IEEE, 681–684

Islam M T, Karunasekera S, Buyya R (2017). dSpark: Deadline-based resource allocation for big data applications in apache spark. In: IEEE 13th International Conference on E-Science. Auckland: IEEE, 89–98

Ivanchenko O, Kharchenko V, Moroz B, Ponochovnyi Y, Degtyareva L (2021). Availability assessment of a cloud server system: Comparing Markov and semi-Markov models. In: 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. Cracow: IEEE, 1–6

Izrailevsky Y, Bell C (2018). Cloud reliability. IEEE Cloud Computing, 5(3): 39–44

Jian C, Ping J, Zhang M (2021). A cloud edge-based two-level hybrid

scheduling learning model in cloud manufacturing. International Journal of Production Research, 59(16): 4836–4850

Levitin G, Xing L, Dai Y (2023). Optimizing partial component activation policy in multi-attempt missions. Reliability Engineering & System Safety, 235: 109251

Li M, Feng J, Xu S X (2023). Toward resilient cloud warehousing via a blockchain-enabled auction approach. Frontiers of Engineering Management, 10(1): 20–38

Li X Y, Liu Y, Lin Y H, Xiao L H, Zio E, Kang R (2021). A generalized petri net-based modeling framework for service reliability evaluation and management of cloud data centers. Reliability Engineering & System Safety, 207: 107381

Liang Y, Lu M, Shen Z M, Tang R (2021). Data center network design for Internet-related services and cloud computing. Production and Operations Management, 30(7): 2077–2101

Lin W, Wang H, Zhang Y, Qi D, Wang J Z, Chang V (2018). A cloud server energy consumption measurement system for heterogeneous cloud environments. Information Sciences, 468: 47–62

Lin W, Wu W, He L (2022). An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers. IEEE Transactions on Services Computing, 15(2): 766–777

Madni S H H, Abd-Latiff M S, Abdullahi M, Abdulhamid S I M, Usman M J (2017). Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. PLoS One, 12(5): e0176321

Malik M K, Singh A, Swaroop A (2022). A planned scheduling process of cloud computing by an effective job allocation and fault-tolerant mechanism. Journal of Ambient Intelligence and Humanized Computing, 13(2): 1153–1171

N'Takpé T, Edgard Gnimassoun J, Oumtanaga S, Suter F (2022). Data-aware and simulation-driven planning of scientific workflows on IaaS clouds. Concurrency and Computation, 34(14): e6719

Niño-Mora J (2019). Resource allocation and routing in parallel multi-server queues with abandonments for cloud profit maximization. Computers & Operations Research, 103: 221–236

Priya V, Sathiya Kumar C, Kannan R (2019). Resource scheduling algorithm with load balancing for cloud service provisioning. Applied Soft Computing, 76: 416–424

Qiu X, Dai Y, Xiang Y, Xing L (2016). A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46(3): 401–412

Qiu X, Dai Y, Xiang Y, Xing L (2019). Correlation modeling and resource optimization for cloud service with fault recovery. IEEE Transactions on Cloud Computing, 7(3): 693–704

Qiu X, Sun P, Dai Y (2021). Optimal task replication considering reliability, performance, and energy consumption for parallel computing in cloud systems. Reliability Engineering & System Safety, 215: 107834

Sayadnavard M H, Toroghi Haghighat A, Rahmani A M (2019). A reliable energy-aware approach for dynamic virtual machine consolidation in cloud data centers. Journal of Supercomputing, 75(4): 2126–2147

Setlur A R, Nirmala S J, Singh H S, Khoriya S (2020). An efficient fault tolerant workflow scheduling approach using replication heuristics and checkpointing in the cloud. Journal of Parallel and Distributed Computing, 136: 14–28

Sharma Y, Si W, Sun D, Javadi B (2019). Failure-aware energy-efficient VM consolidation in cloud computing systems. Future Generation Computer Systems, 94: 620–633

Tian Y, Tian J, Li N (2020). Cloud reliability and efficiency improvement via failure risk based proactive actions. Journal of Systems and Software, 163: 110524

Wang F, Laili Y, Zhang L (2021a). A many-objective memetic algorithm for correlation-aware service composition in cloud manufacturing. International Journal of Production Research, 59(17): 5179–5197

Wang S, Li X, Ruiz R (2020). Performance analysis for heterogeneous cloud servers using queueing theory. IEEE Transactions on Computers, 69(4): 563–576

Wang T, Zhou J, Li L, Zhang G, Li K, Hu X S (2022). Deadline and reliability aware multiserver configuration optimization for maximizing profit. IEEE Transactions on Parallel and Distributed Systems, 33(12): 3772–3786

Wang Y, Zhang L, Yu P, Chen K, Qiu X, Meng L, Kadoch M, Cheriet M (2021b). Reliability-oriented and resource-efficient service function chain construction and backup. IEEE eTransactions on Network and Service Management, 18(1): 240–257

Xu X, Mo R, Yin X, Khosravi M R, Aghaei F, Chang V, Li G (2021). PDM: Privacy-aware deployment of machine-learning applications for industrial cyber–physical cloud systems. IEEE Transactions on Industrial Informatics, 17(8): 5819–5828

Zaloumis C (2022). Are your data centers keeping you from sustainability? Online Article

Zhang C, Kumbhare A G, Manousakis I, Zhang D, Misra P A, Assis R, Woolcock K, Mahalingam N, Warrier B, Gauthier D, Kunnath L, Solomon S, Morales O, Fontoura M, Bianchini R (2021). Flex: High-availability datacenters with zero reserved power. In: ACM/IEEE 48th Annual International Symposium on Computer Architecture. Valencia: IEEE, 319–332

Zhang C, Yao J, Qi Z, Yu M, Guan H (2014). vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming. IEEE Transactions on Parallel and Distributed Systems, 25(11): 3036–3045

Zhang P, Fang J, Yang C, Huang C, Tang T, Wang Z (2020). Optimizing streaming parallelism on heterogeneous many-core architectures. IEEE Transactions on Parallel and Distributed Systems, 31(8): 1878–1896