Cost-optimized Server Allocation and Request Routing in Cloud Systems using Different Service Classes

Niklas Carlsson, *Linköping University*, Sweden
Derek Eager, *University of Saskatchewan*, Canada

***Abstract*—Cloud systems offer scalable and flexible infrastructure but can become expensive if the resource usage is not optimized. To help reduce service delivery costs, this paper presents cost-optimized load-balancing policies for several service classes and uses these to provide valuable system insights. Leveraging a comprehensive system model and considering diverse workload and system parameters, we explore the interplay between resource allocation, request routing, and system performance. Specifically, for each service class, our policies (1) determine the ideal server allocation strategy and (2) identify the most suitable replica site for each incoming request. By deriving optimized policies for different service models and conducting a comparative evaluation, we offer insights into optimal request routing strategies as well as when basic policies are optimal, perform close to optimal, or when more advanced policies are needed. Our findings contribute to improved load-balancing techniques in cloud computing, facilitating enhanced performance and efficiency in distributed environments.**

***Index Terms*—Cloud computing, Edge cloud, Replica servers, Optimization, Server allocation, Request routing, M/M/1, M/D/∞**

I. Introduction

Cloud-based systems are instrumental in the modern computing landscape, offering scalable and flexible infrastructure to meet diverse demands. However, cloud resources can be expensive, making it important to optimize both resource allocation and request routing across cloud service locations.

This paper provides an in-depth analysis and evaluation of cost-optimized load-balancing policies across different service classes. By leveraging a comprehensive system model and considering various workload and system parameters, we provide insights into the intricate interplay between resource allocation, request routing, and overall system performance.

To achieve optimal performance in the context of replicated cloud servers, it is important to strike a balance between delivery times and server resource utilization. Two fundamental decisions underpin this optimization problem: (1) determining the ideal server allocation strategy and (2) identifying the most suitable replica site for each incoming request. These decisions are further complicated by factors such as varying request rates, server startup times, and cloud service costs.

To capture the intricate challenges of load balancing in cloud-based systems, we adopt a holistic approach, consider several service models and system configurations, and derive insights into the relative performance of different load-balancing strategies for these different conditions. Our model assumes that each replica site is running one or more servers, the servers can be turned on and off using deterministic policies based on the current workload conditions and time since the most recent event, requests are forwarded between replica sites based on knowledge about the request rate, cloud service costs are proportional to how much the servers are used, and extra delays are associated with remote service. To span a broad range of use cases, we derived optimized policies for four general service models representing a wide range of cloud service scenarios, varying in how servers are provisioned, whether requests queue, and whether concurrent processing is possible. By comparing the optimized solutions across the service classes, as well as by comparing how well basic load balancing policies perform relative to the optimal solutions of each class, we provide valuable insights into how to best optimize the request routing.

At a high level, the key contributions of our work are:

- **Analysis and Derivation of Optimized Policies:** We present optimized load-balancing policies for different service classes, offering insights into resource allocation and request routing strategies tailored to specific system configurations. To reach these goals, we first derive an exact single-site analysis for each service class (in most cases being the first to do so) and then use the optimal policies derived there to define and prove optimized multi-replica routing solutions.
- **Comprehensive Evaluation and Comparison:** We conduct a thorough evaluation and comparison of load-balancing policies across multiple service classes, highlighting their relative performance and identifying scenarios where certain policies excel or falter. Our (derived) optimal policies allow the performance of simpler and in some cases more practical policies to be presented as relative performance gaps compared to an optimal policy.
- **Insights into System Optimization:** Our analysis yields valuable insights into the optimization of cloud-based systems, shedding light on the intricate interplay between workload characteristics, system parameters, and load-balancing strategies. Of particular value are the insights derived from comparing the best solutions across the four service classes considered here, which emphasize when different basic policies may be optimal, close to optimal, or when it may be better to implement more advanced policies. Although the best policy varies across the parameter space, we found that a hybrid policy, selecting the best from a few basic strategies (e.g., serve locally, direct to the most loaded, or balance evenly) for each workload, often achieves close to optimal performance.

Relation to Prior Work: A subset of our single-site analysis (Section IV) builds on results from our prior work [1], which focused on dynamic server allocation at a single replica under time-varying workloads. In particular, we reuse the analytic results for the M/M/1 and unlimited server models from [1], which serve as components in our broader study. However, the scope and contributions of the present paper are distinct. Here, we introduce a comparative framework across four service models, derive new optimal routing strategies for multi-site systems, and provide insights into how request routing interacts with on-demand resource provisioning across replicas. These extensions are neither addressed nor implied in [1]. For completeness, brief derivations of reused results are provided in the supplementary material, and the full prior work is available via arXiv for transparency.

Overall, our research contributes to advancing the understanding and refinement of load-balancing techniques in cloud-based systems, paving the way for enhanced performance and efficiency in distributed computing environments.

Outline: After presenting our system model (Section II) and the service classes (Section III) considered, Section IV presents our exact single-site analysis (for each service class) and Section V presents the optimal multi-replica routing policies (for each service class) when taking into account transfer costs. We next present numeric comparisons with baseline policies (Section VI) and discuss related works (Section VII), before concluding the paper in Section VIII.

II. System model and service classes

In this section, we describe our system model, the service classes considered, and our workflow. Table I lists our notation.

High-level problem and workflow overview: We consider a cloud-based system comprising multiple replica sites, with the objective of optimizing overall system performance by balancing delivery times and the total server resource usage. Achieving this goal hinges on two critical decisions:

1. determining when it is best to allocate and deallocate server resources at each replica site, and
2. determining the most suitable replica site to direct each incoming request.

The first problem focuses on server allocation while the second focuses on request routing. In this paper, we address both problems jointly. First, we consider four idealized service classes, and derive optimized variations for two of them. Second, we formulate and prove the properties of the optimal request routing strategies for each of these service classes.

Request routing and load assumptions: We consider a system with N server sites and M client locations, and denote the corresponding sets of locations by $\mathcal{N}$ and $\mathcal{M}$. For simplicity, we assume that every client location $c \in \mathcal{M}$ is local to one server location $i^*(c) \in \mathcal{N}$ and remote to all other server locations $i \neq i^*(c), i \in \mathcal{N}$. Here, "local" should be interpreted as the site with the smallest transfer time from the client, and we assume for analytical tractability that all clients have the same minimal "local" transfer time D_L and a uniform "remote" transfer time D_R to all other sites. Furthermore, each client location c has a request rate λ_c and $\lambda_{c,i}$ of this request rate is directed to replica location i. Under these assumptions, each replica site sees a combined load of $\gamma_i = \sum_{c\in\mathcal{M}} \lambda_{c,i}$.

TABLE I
Summary of notation.

Notation	Description
$\mathcal{M}$	Set of client locations.
$\mathcal{N}$	Set of server locations.
λ_c	Requests generated by clients from client location c.
$\lambda_{c,i}$	Request rate from client location c directed to server location i (Note: $\lambda_c = \sum_{i\in\mathcal{N}} \lambda_{c,i}$).
γ_i	Combined request rate directed to server location i. (Note: $\gamma_i = \sum_{c\in\mathcal{M}} \lambda_{c,i}$.)
$i^*(c)$	Server location local to client location c.
λ_c^L	Request rate from client location c that are served locally (i.e., $\lambda_c^L = \lambda_{c,i^*(c)}$).
D_L	Local transfer time (when $c = i^*(c)$).
D_R	Remote transfer time (when $c \neq i^*(c)$).
$R_i = R(\gamma_i)$	Load-dependent response time. Differ between classes.
$C_i = C(\gamma_i)$	Load-dependent cloud-service usage. Differ between classes.
β	Relative cost per time unit that a server is used.
Δ	Average time required to start a server.
S	Average service time.
T	Policy parameter determining when to shut down a server.

Total delivery time: We consider a simple additive model in which the total delivery time associated with a client request is equal to the sum of the transfer time D and the response time R of the server. As noted above, to capture the tradeoff between local and remote service, we assume that the transfer time D_L from a local replica is smaller than the transfer time D_R from a remote replica (i.e., $D_L < D_R$). Furthermore, for each service class, we consider a load dependent response time $R(\gamma_i)$. We call the rate at which total delivery time delay is incurred the request delay cost. Taking the sum over all replicas, the request delay cost is given by:

$$\sum_c \lambda_c^L D_L + \sum_c (\lambda_c - \lambda_c^L) D_R + \sum_i \gamma_i R(\gamma_i), \qquad (1)$$

where $\lambda_c^L = \lambda_{c,i^*(c)}$ is the request rate corresponding to the requests from client location c that are served locally.

Average cloud service costs: Motivated by current cloud costs and most offloaded services being processing-intensive rather than bandwidth intensive, we ignore network costs (assumed much smaller than processing costs under the scenarios of interest here). Summing over all replica locations, the total cloud cost is then:

$$\beta \sum_i C(\gamma_i), \qquad (2)$$

where $C(\gamma_i)$ is the fraction of time server i is active (including setup times, busy times, and delayed off periods) and β is its relative cost per time unit the server is used. Here, the parameter β allows any desired relative weightings in importance between request delay cost and cloud service cost.

Allocation times and de-allocation polices: We foresee a system in which each server instance takes some time Δ to start up (when requests are made to a server replica that currently does not have an active VM for the service) and that the service provider only pays for the VMs when they are in use (i.e., either are starting up or running). Two important aspects to consider when considering the total system cost are (1) the extra time that a request needs to wait while a new

server instance (regardless of service model) is allocated (e.g., a server or VM is started up) and (2) the time that it takes to de-allocate a server (or VM). Startup times are variable in practice, and we model these as exponentially distributed with mean Δ. Furthermore, shutdown policies would typically be deterministic in practice, and so we assume that de-allocation times are deterministic with shut-down times T.

Combined optimization problem: For each of the above service classes, we derive optimized request routing solutions that balance the two competing objectives. For this purpose, use a combined objective function that uses a factor β to weigh the tradeoff between the total request delay cost (incorporating response time due to data processing + network delays) and the total cost of running the replica servers. In particular, we aim to minimize:

$$\sum_c \lambda_c^L D_L + \sum_c (\lambda_c - \lambda_c^L) D_R + \sum_i \gamma_i R(\gamma_i) + \beta \sum_i C(\gamma_i). \tag{3}$$

Now, recognizing that we can restructure some terms (i.e., break out γ_c in the sums instead of λ_c; here leveraging that the total load $\sum_c \lambda_c = \sum_i \gamma_i$ can be rewritten as $\sum_c \gamma_c$ under our assumption that every client location c is local to one server location $i^*(c)$) and that we can subtract D_L from every request's delay without impacting the optimal solution (since the subtracted sum $\sum_c \gamma_c D_L$ is independent of to which replica c that each request is directed), we rewrite the objective function (after this subtraction) as:

$$\sum_c (\gamma_c - \lambda_c^L) D'_R + \sum_i \gamma_i R(\gamma_i) + \beta \sum_i C(\gamma_i), \tag{4}$$

where $D'_R = (D_R - D_L)$. To see this, note that $\lambda_c^L D_L + (\gamma_c - \lambda_c^L) D_R - \gamma_c D_L = (\gamma_c - \lambda_c^L)(D_R - D_L) = (\gamma_c - \lambda_c^L) D'_R$. While these restructurings of the first term in the objective function were done to better fit with the simplest interpretations of later derivations, we note the first sum of Equation (4), using the same arguments, also can be written and interpreted as $\sum_c (\lambda_c - \lambda_c^L) D'_R$. For the reminder of our multi-site analysis, we will use Equation (4) as our primary objective function.

Remark on locality assumptions: While the one-to-one mapping (that each client is local to one serve site) and the uniform remote access penalty (D_L and D_R delays) may not fully reflect all real-world scenarios—where clients can be similarly close to multiple replicas or distances may be heterogeneous—it serves as a useful abstraction. These assumptions simplify the analysis, enabling us to derive provable structural properties based on closed-form expressions that would be intractable under more general proximity models.

We emphasize that "local" in our model reflects the site with minimal expected transfer time, not necessarily geographic proximity. For tractability, we assume homogeneous local and remote transfer times across clients, which enables derivation of closed-form expressions and optimal routing strategies. This abstraction makes it possible to analyze fundamental tradeoffs between latency and resource usage. Exploring more detailed models (e.g., heterogeneous client distributions or multiple low-latency options) would be a valuable direction for future work, particularly for understanding policy behavior in scenarios with ambiguous proximity.

Remark on processing-oriented workloads and generalized remote penalty modeling: Our focus is on services that are primarily compute-bound or subject to startup overheads, common in modern cloud environments such as serverless computing, ML inference platforms, or analytics workloads. In such systems, response time and cost are often dominated by compute delays, queuing, or cold-start latency, rather than data transfer volume. However, while we do not explicitly model bandwidth-based pricing or network congestion, the remote delay penalty term D_R can also capture a broader range of real-world cost or performance impact associated with remote service. For example, in addition to added latency, this abstraction may reflect provisioning overhead, cross-region transfer costs, or policy-driven constraints. The model is thus flexible: depending on deployment context, D_R may represent latency, monetary cost, or both. This generality allows our framework to reflect realistic tradeoffs for a variety of geo-distributed scenarios while keeping the analysis tractable.

III. Service Classes

Within the above system model, we consider four basic service classes. Each model represents a different cloud service paradigm, balancing tractability with practical relevance. The models vary in how servers are provisioned, whether requests queue, and whether concurrent processing is possible. These distinctions are central to the performance and cost tradeoffs studied in later sections.

- **Individual service:** With this service class, a separate server instance (e.g., VM or container) is dynamically allocated for each new request and then released when the request is completed. This service class do not allow queuing and captures serverless or traditional Function-as-a-Service (FaaS) environments with strict one-request-per-instance behavior [2], [3].[1]
- **Dynamic M/M/1 server:** With this service class, each replica location is running a single queue-based M/M/1 server. To reduce service costs, we assume a scenario in which the server can be dynamically de-allocated when there are no requests being served and then re-allocated when new requests arrive. Unlike *Individual Service*, requests may queue if the server is busy. This service class models single-threaded VM-based services with startup overhead and latency-sensitive queuing.
- **Reactive Unlimited server system:** With this service class, new server instances are spun up reactively, with the goal of always matching the number of active servers to the number of requests in the system. To do so, we (1) initiate allocation of a new server instance whenever a new request arrives while there is not a new server allocation already in progress, (2) constrain the total number of servers (active or in process of being allocated)

[1]While many FaaS platforms historically enforce strict one-request-per-instance behavior, modern "Serverless 2.0" systems increasingly support concurrency within a container and allow for provisioned warm instances [4], [5], in some cases making them better captured by our M/D/∞ service class.

to be at most equal to the number of requests currently in the system, and (3) cancel ongoing allocations and de-allocate servers as needed to adhere to this constraint. Unlike *Dynamic M/M/1*, this class allows multiple concurrent servers, and the class differs from *Individual Service* by allowing short queuing during startup and by not requiring one-to-one request-server mapping.

- **Dynamic M/D/∞ server:** With this service class, each replica location is running a single M/D/∞ server that can be dynamically de-allocated and re-allocated. The use of an M/D/∞ model, means that a server can process all incoming requests in parallel without queuing. This model captures highly parallel systems such as stateless microservices or cloud APIs with high concurrency support (e.g., default container concurrency of 100 in serverless platforms [4]), or autoscaled systems like Google Cloud Run [6], IBM Code Engine [7], and Azure Functions [5].

Why these models? These four classes span a wide spectrum of realistic cloud behaviors: from strictly serialized (M/M/1) to infinitely parallel (M/D/∞); from stateless per-request allocation (*Individual*) to responsive scaling (*Reactive Unlimited*). They capture key operational modes found in practice—such as serverless computing, virtual machine hosting, microservices, and autoscaling APIs—while allowing for closed-form analysis.

We selected these four service models because they collectively represent a diverse and practically relevant cross-section of cloud deployment strategies, yet remain analytically tractable. This tractability is essential for deriving provably optimal request-routing and resource-allocation policies, and for enabling head-to-head comparisons with simpler or heuristic strategies. In contrast, more general models such as G/G/k offer broader modeling flexibility, but they typically lack closed-form expressions and are analytically intractable, making them unsuitable for the kind of structural analysis and optimality proofs we pursue in this paper.

For example, even the M/M/1 expressions used in this paper were not previously available and were derived in our earlier work [1] (summarized in the appendix). By focusing on models that support such derivations, we strike a balance between realism and rigor—yielding both actionable system insights and provable guarantees across a range of cloud service scenarios.

Relation to real-world systems: The service models studied here abstract key mechanisms used in real-world cloud platforms such as AWS, Azure, and Google Cloud. For example, serverless computing platforms like AWS Lambda or Azure Functions [2], [5] often use per-request instance allocation, though recent extensions (termed "Serverless 2.0" [4]) support concurrent handling and warm container pools. VM-based services use keep-alive policies (captured by *Dynamic M/M/1*) to reduce idle costs [3]. Orchestrated microservice platforms such as OpenWhisk [8], Knative [9], or Kubernetes-based autoscalers follow reactive resource-matching behavior similar to our *Reactive Unlimited* or M/D/∞ models. While commercial platforms introduce additional complexity (e.g., batching, cold-start suppression [10], predictive scaling), our simplified models capture the essential tradeoffs in queuing, concurrency, and cost—making them broadly representative of real deployments while remaining analytically tractable.

IV. Exact Single-site Analysis

In this section, we derive exact single-site expressions for each service class and derive optimal shut-down policy thresholds for the service models for which this is applicable (i.e., *Dynamic M/M/1* server and *Dynamic M/D/∞* server). For the single-site case, our model reduces to $\gamma R(\gamma) + \beta C(\gamma)$.

A. Individual Service

This service class can be seen as a function-as-a-service type of approach without any caching of server process instances. Therefore, each service instance will see a startup-period of Δ and an average service time S, resulting in an average response time $R = \Delta + S$ and average cloud service usage $C = \gamma(\Delta + S)$, where C simply equals the average number of servers that are active or in startup as calculated using Little's law.

Combined single-replica cost: Now, combining these costs, the total single-replica cost can be calculated as:

$$\gamma(1+\beta)(\Delta + S). \tag{5}$$

B. Dynamic M/M/1 Server

As shown in [1] and outlined in Appendix B, we can derive exact closed-form expressions for both the average response time R and the average cloud service usage C for an M/M/1 server with Erlang distributed server de-allocation times (with Erlang shape parameter k), and then letting $k \to \infty$ to obtain the deterministic shut-down policy results.[2] In this case:

$$R = \frac{S}{1-\gamma S} + \frac{\Delta(1+\gamma\Delta)}{e^{\gamma T} + \gamma\Delta}, \quad C = 1 - \frac{1-\gamma S}{e^{\gamma T} + \gamma\Delta}. \tag{6}$$

Singe-replica optimization: Consider now the optimal T when γ is known and stable. In this case, the combined cost function has the derivative:

$$\frac{d}{dT}(\gamma R + \beta C) = -\frac{\gamma e^{\gamma T}}{(e^{\gamma T} + \gamma\Delta)^2}\left((\gamma\Delta(1+\gamma\Delta) - \beta(1-\gamma S)\right). \tag{7}$$

We note that the sign of the derivative depends on γ. Solving for the γ^* for which the derivative is zero (and changes sign), we find only one positive root to the equation:

$$\gamma^* = \frac{\sqrt{4\beta\Delta^2 + (\beta S + \Delta)^2} - (\beta S + \Delta)}{2\Delta^2}. \tag{8}$$

Keeping track of the sign, we note that the derivative is non-negative whenever $0 < \gamma \leq \gamma^*$ and negative otherwise. Given these properties, the objective function is always non-decreasing (with T) when $0 < \gamma \leq \gamma^*$ and non-increasing (with T) for $\gamma^* < \gamma$. Therefore, it is optimal to use $T = 0$ when $\gamma \leq \gamma^*$ and $T = \infty$ otherwise.

[2] In [1] the definition of "C" differs slightly from that used here, resulting in expressions that differ by a factor of S.

Key observation (M/M/1 case): When γ is known and stable, it is optimal to use $T = 0$ when $\gamma \leq \gamma^*$, where $\gamma^* = \frac{\sqrt{4\beta\Delta^2+(\beta S+\Delta)^2}-(\beta S+\Delta)}{2\Delta^2}$, and $T = \infty$ otherwise.

Combined single-replica cost: Now, combining these costs, the minimized total single-replica cost becomes:

$$\begin{cases} \frac{\gamma S}{1-\gamma S} + \gamma\Delta + \beta\gamma\frac{\Delta+S}{1+\gamma\Delta}, & \text{if } \gamma \leq \gamma^* \\ \frac{\gamma S}{1-\gamma S} + \beta, & \text{otherwise.} \end{cases} \tag{9}$$

C. Reactive Unlimited Server System

As derived in [1] and outlined in Appendix A, this service class has a response time and average cloud service usage of

$$R = \Delta + S, \quad C = \gamma(S + \frac{\Delta}{1+\gamma\Delta}). \tag{10}$$

The response time is equivalent to using a separate server per request, but with lower server usage. This is explained by the efficiency that results from taking a newly-free existing server for a waiting request instead of always requiring a new server.

Combined single-replica cost: Combining the costs, the total single-replica cost can be calculated as:

$$\gamma(\Delta + S) + \beta\gamma\left(S + \frac{\Delta}{1+\gamma\Delta}\right). \tag{11}$$

D. Dynamic M/D/∞ Server

For this analysis, we consider a renewal period that starts (and ends) with the initial request triggering a new setup period. Considering such a renewal period, we recognize that (1) the one request triggering the setup period will endure a waiting time of $S+\Delta$, (2) there will be on average $\gamma\Delta$ requests per renewal period that will arrive during the setup period, (3) these requests on average will endure an extra waiting time of Δ,[3] and (4) the rest of the requests arriving during a renewal period arrive after setup and can be served as soon as they arrive (i.e., only enduring the service time S).

To calculate how long a busy period is (including the delayed off periods), we note that such time period can be modeled using an inflated service time of $S+T$. In this case, the average busy period $E[B]$ is $\frac{1}{\gamma}(e^{\gamma(S+T)}-1)$. Given these observations, the average response time can be calculated as:

$$R = S + \frac{(1+\gamma\Delta)\Delta}{1+\gamma\Delta+\gamma E[B]} = S + \frac{(1+\gamma\Delta)\Delta}{\gamma\Delta+e^{\gamma(S+T)}}. \tag{12}$$

Furthermore, the fraction of time that the server is either in setup phase (time Δ), actively servicing requests, or in delayed turn-off mode can easily be calculated as:

$$C = \frac{\Delta + E[B]}{1/\gamma + \Delta + E[B]} = \frac{\gamma\Delta + e^{\gamma(S+T)} - 1}{\gamma\Delta + e^{\gamma(S+T)}}. \tag{13}$$

Note that the above analysis relies on $S+T$ being deterministic. In this case, even when there are m clients in the system when the busy period starts, one can consider it as if there is only a single client that starts the busy period. In contrast, in the case that S would be variable, then the "first" request would need to be considered as the one with the maximum S of all m requests in the system at that time. However, in this case no exact analysis is easily attainable.

Single replica optimization: Similar as for the M/M/1 service class, we can obtain the optimized policy by considering the derivative of the combined cost function:

$$\frac{d}{dT}(\gamma R + \beta C) = -\frac{\gamma(\gamma\Delta(1+\gamma\Delta)-\beta)e^{\gamma(S+T)}}{(\gamma\Delta+e^{\gamma(S+T)})^2}. \tag{14}$$

From this expression, it is then easy to see that $\frac{d}{dT}(\gamma R + \beta C) \geq 0$ whenever $\beta \geq \gamma\Delta(1+\gamma\Delta)$ and $\frac{d}{dT}(\gamma R + \beta C) \leq 0$ otherwise. The point where the derivative (and hence the optimal T being used) changes happens when $\gamma^2\Delta^2 + \gamma\Delta - \beta = 0$, which only has one (positive) solution $\gamma^* = \frac{\sqrt{4\beta+1}-1}{2\Delta}$ (and $\Delta \neq 0$). It is therefore optimal to use $T = 0$ when $\gamma < \gamma^*$ and $T \to \infty$ otherwise.

Key observation (M/D/∞ case): When γ is known and stable, it is optimal to use $T = 0$ when $\gamma \leq \gamma^*$, where $\gamma^* = \frac{\sqrt{4\beta+1}-1}{2\Delta}$, and $T = \infty$ otherwise.

Combined single-replica cost: Combining the above results, we find that the minimized total delivery cost associated with a single replica can be calculated as:

$$\begin{cases} \gamma\left(S + \frac{(1+\gamma\Delta)\Delta}{\gamma\Delta+e^{\gamma S}}\right) + \beta\frac{\gamma\Delta+e^{\gamma S}-1}{\gamma\Delta+e^{\gamma S}}, & \text{if } \gamma < \gamma^*, \\ \gamma S + \beta, & \text{otherwise.} \end{cases} \tag{15}$$

E. Numeric Single-Replica Comparison

Throughout the paper, without loss of generality, we normalize units by setting the average service time $S = 1$. With this choice, all time-based quantities (e.g., startup time Δ, response time R, and server usage C) are measured relative to the time required to serve a single request. This allows the request rate γ to be interpreted as the average number of arrivals per service time unit, which corresponds to server utilization in the M/M/1 case. The cost function combines expected response delay and server usage, both expressed in normalized time units. The parameter β controls their relative importance: for example, setting $\beta = 1$ assigns equal cost to one unit of delay and one unit of server usage. This normalization is standard in analytical modeling and enables us to highlight the tradeoff structure while maintaining generality.

High-level service class comparison: Figure 1 compares the total cost for the four service classes using the default case of $\Delta = 1$ and $\beta = 1$. We observe a clear cost ordering: the *Dynamic M/M/1* class saturates at $\gamma = 1$ as utilization approaches full capacity, while the other classes maintain acceptable response times beyond this point. This is achieved by either starting new servers as more requests come in (e.g., with *Individual Service* and *Reactive Unlimited*) or through use of servers able to serve multiple requests in parallel without inflating response times (i.e., *Dynamic M/D/*∞). Of these service classes, the *Dynamic M/D/*∞ class (which may

[3]Note: If considering fixed (rather than exponential) setup period, then the $\gamma\Delta$ arrivals that (on average) arrive during the setup period have an average extra waiting time of $\Delta/2$ (rather than Δ).

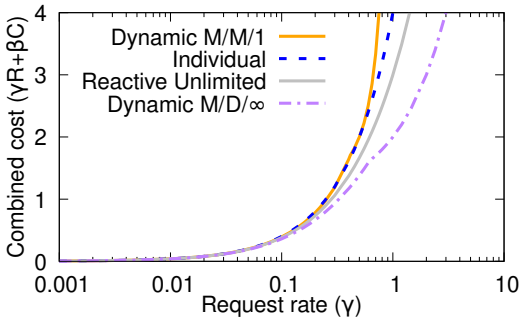
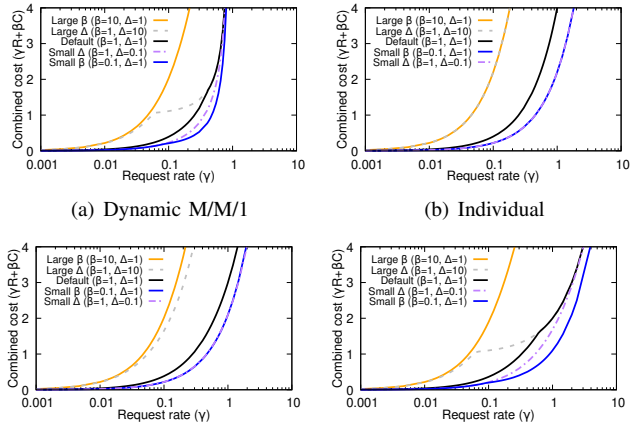


Fig. 1. High-level service class comparison using default normalized values $S = 1$, $\Delta = 1$, and $\beta = 1$, where costs are expressed in units of normalized time. **Observation:** The high separation between at what requests rates (x-axis, shown on log axis) that the combined costs (y-axis) spikes for with *Dynamic M/M/1* and *Dynamic M/D/*∞ illustrates the wide span of service classes considered. We also note that the other two service classes (*Individual* and *Reactive Unlimited*) provide interesting intermediate service categories.



(a) Dynamic M/M/1 (b) Individual

(c) Reactive unlimited (d) Dynamic M/D/∞

Fig. 2. Relative impact of high/low startup times (Δ) and high/low cloud-service cost weights (β) on total cost, using normalized units with $S = 1$. Costs are plotted in normalized time units for each service class. **Observation:** The discontinuity observed for both dynamic service classes (M/M/1 and M/D/∞) for the $\Delta = 10$ curves captures the operation point where the optimal policy of each of these classes switches from using $T = 0$ to always keeping the servers on (i.e., using $T \to \infty$).

be reflective of large multicore servers, for example) would provide the most attractive cost tradeoff assuming a fixed value of β, and the *Reactive Unlimited* class is seen as a significant improvement over the more naive *Individual Service* class, as it avoids de-allocating servers when there are waiting requests (i.e., servers can be reused).

Relative impact of parameters: We next consider the impact of high vs. low startup times (Δ) and high vs. low cloud-service cost weights (β). Figure 2 summarizes these results. We note that a large cloud-service cost (e.g., $\beta = 10$ curves) relatively consistently has the most adverse effect on the optimal costs of the four services. For both dynamic service classes (M/M/1 and M/D/∞) there is a clear discontinuity for the $\Delta = 10$ curves. This captures the operation point where the optimal policy of each of these classes switches from using $T = 0$ to always keeping the servers on (i.e., using $T \to \infty$). Otherwise, for these service classes, the startup time Δ seems to have a relatively smaller impact than the cloud-service cost weight β, as a factor 10 up or down on Δ impacts the total cost less than applying the same factor to β.

V. Multi-Replica Routing with Transfer Cost

Let us now consider the full cost model in Equation (4). For this analysis, without loss of generality, we assume that the replicas are enumerated from the replica with the biggest to the smallest local request rate; i.e., $\lambda_1 > \lambda_2 > ... > \lambda_N$.

A. Service-Independent Optimality Property

Before deriving the optimal request routing strategies for each specific service class, we begin by establishing a basic monotonicity property that holds independently of the service model. Although this result is intuitive—clients with higher local demand should be assigned higher load—it serves as a useful foundation for proving more complex properties in later sections In particular, we show that there exists an optimal solution in which $\gamma_i \geq \gamma_j$ whenever $\lambda_i > \lambda_j$. This is formalized and proved in the following theorem.

Theorem 1. *There exists an optimal solution to the load allocation problem in which* $\gamma_i \geq \gamma_j$ *whenever* $\lambda_i > \lambda_j$.

Proof. Assume to the contrary that there is no optimal solution that satisfies this criterion. In this case, in an optimal solution there must exist at least one pair of replicas, say a and b, where $a < b$, such that $\gamma_a < \gamma_b$ (and $\lambda_a > \lambda_b$). Now, consider an alternative solution in which $\gamma'_a = \gamma_b$ and $\gamma'_b = \gamma_a$. Since the total load γ_i of these two replicas are only switched, this solution would clearly have the same average response time (i.e., $\frac{1}{\sum_i \gamma_i} \sum_i \gamma_i R_i$) and cloud service usage (i.e., $\sum_i C_i$). Any difference between the cost of the two solutions must therefore be in the amount of remote service (i.e., $\gamma_a - \lambda_a^L + \gamma_b - \lambda_b^L$). In fact, due to the equal switch in load, the only difference is in how large a fraction of the requests is served locally (i.e., $\lambda_a^L + \lambda_b^L$), where more local service is desirable.

Now, consider the local service with the new allocation:

$$\begin{aligned} \lambda_a^{L\prime} + \lambda_b^{L\prime} &= \min[\lambda_a, \gamma'_a] + \min[\lambda_b, \gamma'_b] \\ &= \min[\lambda_a, \gamma_b] + \min[\lambda_b, \gamma_a] \\ &\geq \min[\lambda_a, \gamma_a] + \min[\lambda_b, \gamma_b] \quad = \lambda_a^L + \lambda_b^L \end{aligned} \tag{16}$$

Here, we use the fact that $\min[X, Y] + \min[x, y] \geq \min[X, y] + \min[x, Y]$ whenever we have that $X \geq x$ and $Y \geq y$. This shows that the modified solution is no worse than the original solution. Since this process can be repeated until we have a solution that satisfies the criteria of strictly ordered loads (i.e., $\gamma_i \geq \gamma_j$ for all $i < j$) we have shown that there exists at least one solution that is no worse than the original solution (assumed optimal) that satisfies the claimed criteria. This contradicts the assumption and completes the proof. □

B. Optimality when Individual Service

When each client is served individually, there is no benefit in forwarding requests remotely. To see this, we note that both the response time and the per-request processing overhead is location independent. With all requests being served locally, the cost associated with each replica is given by Equation (5), and the total delivery cost can therefore be calculated as:

$$\sum_{i=1}^{N} \lambda_i (1 + \beta)(\Delta + S). \tag{17}$$

C. Optimality with Dynamic M/M/1 Servers

To characterize the optimal solution, we prove properties regarding (1) the server policies used by the different replicas and (2) the load split across replicas. At a high level, the structure is based on the identification of three groups of replicas with different service policies, followed by careful load balancing within and across these groups. The first theorem below defines the three groups and their operation.

Theorem 2. *There exists an optimal solution that satisfies the properties of Theorem 1 with the following structure:*

- *Replicas* $1 \leq i \leq a$*:* $T_i \to \infty$*,* $R_i = \frac{S}{1-\gamma_i S}$*,* $C_i = 1$*,*
- *Replicas* $a < i \leq b$*:* $T_i = 0$*,* $R_i = \frac{S}{1-\gamma_i S} + \Delta$*,* $C_i = \frac{\gamma_i(\Delta+S)}{1+\gamma_i\Delta}$*,*
- *Replicas* $b < i \leq N$*:* $\gamma_i = 0$*,* $C_i = 0$*,*

Proof. First, consider each replica in isolation. From the key observation of Section IV-B, we know that for any replica with a known and non-zero load γ_i it is optimal to either use $T_i = 0$ or $T_i \to \infty$. Therefore, given a non-zero load γ_i, a replica is best operated using one of these two operation points. Second, any replica with no load (i.e., $\gamma_i = 0$) should be turned off completely (i.e., $C_i = 0$). Given these two observations, we must only show (1) that there exists an optimal solution in which we do not have any cases for which $T_i = 0$ and $T_j \to \infty$ for some $i < j$ with $\gamma_i < \gamma_j$ and (2) that only the replicas with the lowest local load λ_i have zero load.

To prove the first property, assume to the contrary that there exist two replicas $i < j$ such that replica i uses $T_i = 0$ and $T_j \to \infty$. In this case, it is easy to see that it would be better to change so that $T_i \to \infty$ and $T_j = 0$. First, note that the change in cost for these two entities are $\gamma_i\Delta - \beta\frac{1-\gamma_i S}{1+\gamma_i\Delta}$ and $\gamma_j\Delta - \beta\frac{1-\gamma_j S}{1+\gamma_j\Delta}$, respectively, but with opposite signs. Second, note that the derivative of this function is non-negative; i.e., $\frac{d}{d\gamma}(\gamma\Delta - \beta\frac{1-\gamma S}{1+\gamma\Delta}) = \frac{\beta\Delta+\beta S+\Delta^3\gamma^3+2\Delta^2\gamma+\Delta}{(1+\gamma\Delta)} \geq 0$. Therefore, it is always better to change so that replica i uses $T_i \to \infty$ and replica j uses $T_j = 0$.

Finally, to prove the second property and complete the proof, we again use proof by contradiction. Here, we can simply assume that (to the contrary) the optimal solution has a pair of replicas i and j such that $i < j$ and $\gamma_i = 0$ and $\gamma_j > 0$. However, as per Theorem 1, there exists a solution with no lower cost that satisfies our conditions (that can be obtained by simply switching the loads for the two replicas). We note that replicas with $\gamma_i = 0$ simply correspond to a special case where some number of replicas (in our case the last $N - b$ replicas) serve no requests. Furthermore, for this case, it clearly is optimal to never turn on the server (i.e., $C_i = 0$ for these replicas). This completes the proof. □

While the third group thus far can be seen as a special case of the second group, we have kept it separately as it is operated in a different manner than the replicas in the other groups and it (as we will soon see) allows us to much more easily characterize the load distributions within each group. However, before looking closer at this, we note that the identification of the three groups provides a high-level solution approach to finding the optimal solution.

At a high level, the idea is to search over all group splits, where indexes a and b determines the exact splits. This would involve considering $\mathcal{O}(N^2)$ cases, where for each case the cost of the optimal solution could be calculated as follows:

$$\sum_{i=1}^{a}\left(\frac{\gamma_i S}{1-\gamma_i S}\right) + \sum_{i=a+1}^{b}\left(\frac{\gamma_i S}{1-\gamma_i S} + \gamma_i\Delta\right) + \beta a + \beta\sum_{i=a+1}^{b}\frac{\gamma_i(\Delta+S)}{1+\gamma_i\Delta} + D_R\sum_{i=1}^{b}(\gamma_i - \lambda_i^L). \quad (18)$$

Now, as noted above, there are still several more things that must be determined before having a full characterization of the optimal solution. In particular, we still need to determine how much load each replica should serve. To determine how the loads are split within each of these groups (beyond what is implied by Theorem 1) some additional care is therefore needed here. We next provide some key load-distribution insights into the structural properties of the optimal solution within the first two groups, followed by a discussion on how an optimal solution that satisfies all the identified properties effectively can be identified.

Theorem 3. *An optimal solution satisfies the following properties. First, there exists two load limits* $\underline{\gamma_a}$ *and* $\overline{\gamma_a}$ *associated with replicas* a' *and* a'' *(*$a' < a''$*) such that* $\underline{\gamma_a} \leq \overline{\gamma_a}$ *and*

(1) $\gamma_i = \overline{\gamma_a} \leq \lambda_i$ *for* $1 \leq i \leq a'$*,*
(2) $\gamma_i = \lambda_i$ *for* $a' < i \leq a''$*, and*
(3) $\gamma_i = \underline{\gamma_a} > \lambda_i$ *for* $a'' < i \leq a$*.*

Second, there exists two load limits $\underline{\gamma_b} \leq \overline{\gamma_b}$ *associated with two replicas* b' *and* b'' *(*$b' < b''$*) such that*

(4) $\gamma_i = \overline{\gamma_b} \leq \lambda_i$ *for* $a < i \leq b'$*,*
(5) $\gamma_i = \lambda_i$ *for* $b' < i \leq b''$*, and*
(6) $\gamma_i = \underline{\gamma_b} > \lambda_i$ *for* $b'' < i \leq b$*.*

Proof. We will prove this by contradiction, assuming that there exists an optimal solution that does not satisfy the above property (but Theorem 1).

High-level approach: We start by considering the replicas with index $i \leq a$ in isolation (showing that an improved solution can be obtained that satisfies the above properties for those replicas), then the replicas with index $a < i \leq b$ in isolation, and lastly show that the finally constructed (and improved) solution satisfies all claimed properties.

First group ($i \leq a$): First, consider the replicas with index $i \leq a$. For these replicas, we can break up the cost into two parts: (1) the response times (including queuing delays) and (2) the remote access costs. Now, consider the response time terms (i.e., $f = \gamma_i R_i = \frac{\gamma_i S}{1-\gamma_i S}$). This function is convex, with both positive derivative and second derivative (i.e, $\frac{df}{d\gamma} > 0$ and $\frac{d^2 f}{d\gamma^2} > 0$). For this reason, $f(\gamma+x)+f(\gamma-x)$ is monotonically increasing with x, and as long as load balancing does not impact the remote access cost, it is always desirable for a partial solution consisting of two replicas with request rates $\gamma + x$ and $\gamma - x$ to be as load-balanced as possible (i.e., small x). Now, let us use this observation to prove the optimal load characteristics for the replicas with index $i \leq a$.

For this part of the proof, we will first prove that there exists an optimal solution for which all replicas with $\gamma_i < \lambda_i$

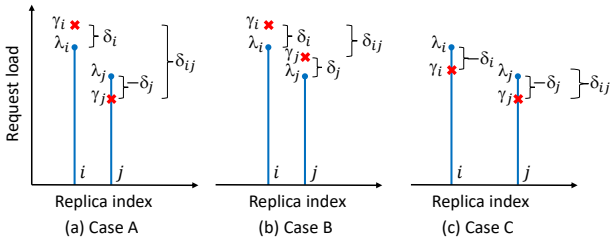


Fig. 3. Potentially violated example cases that must be considered in the proof of Theorem 3. Intuitively, in case A some degree of offloading can be done from the more loaded to the less loaded server, so to improve load balancing, effectively reducing the average response times, without increasing the fraction of remote service. In cases B and C, some degree of load balancing is possible without impacting the overall level of remote (or local) service.

(if any) have index ahead of those with $\gamma_i = \lambda_i$ (if any), which have index lower than those with $\gamma_i > \lambda_i$ (if any). We will prove this by contradiction, assuming that there exists an optimal solution that does not satisfy the above property (but Theorem 1). Now, we can show that for any pair of consecutively ordered replicas i and j (where $j = i + 1$) for which the property does not hold, we can improve the "optimal" solution by creating a solution that satisfies the above property (and Theorem 1). To do this, we must consider all three pairwise cases that violate the property. These three cases are illustrated in Figure 3. For simplicity we will also use the notation used in the figure in the proof of each case. Here, we let $\delta_i = \gamma_i - \lambda_i$, $\delta_j = \gamma_j - \lambda_j$, and define the cases based on the signs of δ_i and δ_j. Furthermore, we let $\delta_{ij} = \gamma_i - \gamma_j$.

Case A ($\delta_i > 0$ and $\delta_j < 0$): For this case, an improved solution can always be achieved by moving $\min(\delta_i - \delta_j, \delta_{ij}/2)$ of the request load from replica i to replica j. To see this, consider the two subcases:

- When $\delta_i - \delta_j < \delta_{ij}/2$, then there are (1) no change of the fraction of requests processed remotely (since replica i after the change directs δ_j remotely and replica j serves δ_i more than its local load) and (2) the load is more balanced (since we moved load from the more loaded replica to the less replica, but not past the balancing point). Therefore, due to the above-mentioned properties of $f(x)$, we see a cost reduction. Furthermore, it is easy to see that (3) $\gamma_i^{new} \geq \gamma_j^{new}$ (since the move is smaller than δ_{ij} which would balance the two), and (4) $\gamma_i^{new} = \lambda_i - \delta_j < \lambda_i$ and $\gamma_j^{new} = \lambda_j + \delta_i \geq \lambda_j$. The new (and improved) solution therefore satisfies our claimed properties.
- When $\delta_{ij}/2 \leq \delta_i - \delta_j$, then there is (1) a reduction in the fraction of request processed remotely and (2+3) the load becomes equalized between the replicas (i.e., $\gamma_i^{new} = \gamma_j^{new}$). Therefore, there is a clear cost reduction. Furthermore, we note that (4) either $\gamma_i^{new} < \lambda_i$ (in which case it does not matter if $\gamma_j^{new} \geq \lambda_j$ or not) or $\gamma_i^{new} \geq \lambda_i$ (in which case $\gamma_j^{new} > \lambda_j$).

Case B ($\delta_i > 0$ and $\delta_j > 0$): For this case, an improved solution can always be achieved by moving $\min(\delta_i, \delta_{ij}/2)$ of the request load from replica i to replica j. To see this, consider the two subcases:

- When $\delta_i < \delta_{ij}/2$, then there are (1) no change of the fraction of requests processed remotely (since both γ_i^{new} and γ_j^{new} remain no smaller than λ_i and λ_j, respectively) and (2) the load is more balanced (since we moved load from the more loaded replica to the less replica, but not past the balancing point). Therefore, due to the above-mentioned properties of $f(x)$, we see a cost reduction. Furthermore, it is easy to see that (3) $\gamma_i^{new} \geq \gamma_j^{new}$, and (4) $\gamma_i^{new} = \lambda_i$ and $\gamma_j^{new} \geq \lambda_j$.
- When $\delta_{ij}/2 \leq \delta_i$, then there are (1) no change of the fraction of requests processed remotely and (2+3) the load becomes equalized between the replicas (i.e., $\gamma_i^{new} = \gamma_j^{new}$). Therefore, there is a clear cost reduction. Finally, it is easy to see that (4) $\gamma_i^{new} > \lambda_i$ and $\gamma_j^{new} > \lambda_j$).

Case C ($\delta_i < 0$ and $\delta_j < 0$ and $\delta_{ij} > 0$): For this case, an improved solution can always be achieved by moving $\min(-\delta_j, \delta_{ij}/2)$ of the request load from replica i to replica j. To see this, consider the two subcases:

- When $-\delta_j < \delta_{ij}/2$, then there are (1) no change of the fraction of requests processed remotely (since both γ_i^{new} and γ_j^{new} remain no greater than λ_i and λ_j, respectively) and (2) the load is more balanced (since we moved load from the more loaded replica to the less replica, but not past the balancing point). Therefore, due to the above-mentioned properties of $f(x)$, we see a cost reduction. Furthermore, it is easy to see that (3) $\gamma_i^{new} > \gamma_j^{new}$ and (4) $\gamma_i^{new} < \lambda_i$ and $\gamma_j^{new} = \lambda_j$.
- When $\delta_{ij}/2 \leq -\delta_i$, then there are (1) no change of the fraction of requests processed remotely and (2) the load becomes fully balanced between the two replicas. Therefore, there is a clear cost reduction. Furthermore, it is easy to see that (3) $\gamma_i^{new} = \gamma_j^{new}$ and (4) $\gamma_i^{new} < \lambda_i$ and $\gamma_j^{new} < \lambda_j$).

We note that in all cases that violate the properties of our claimed optimal solution, a better solution can be obtained by "fixing" cases that do not satisfy the claimed property until we have a solution that does. Furthermore, it should be noted that the above algorithm always pushes load left-to-right (i.e., from i to $j = i + 1$) and that the new solution is obtained in a finite number of steps.

Next, we show that all replicas with load $\gamma_i \leq \lambda_i$ (if any) should be load balanced (i.e., the most loaded replicas in this group) and all replicas with $\gamma_i > \lambda_i$ (if any) should be load balanced (i.e., the least loaded replicas in this group). This can easily be shown (and such a solution obtained) thanks to the convexity of the function $f(x)$, which ensures that the average service times across all the replicas within each of the two groups are minimized by load balancing them, without impacting the fraction of service obtained remotely. We now have an improved solution that satisfies all our claimed properties for replicas $1 \leq i \leq a$, completing our proof for the first group (in isolation).

Second group ($a < i \leq b$): Second, we basically repeat the above proof for the second group of replicas (in isolation); i.e., the replicas labeled replicas $a < i \leq b$. The proof for this subset follows the exact same steps as for the first group. The only difference lies in the cost terms not associated with the remote processing, which now reads as $g = \frac{\gamma_i S}{1-\gamma_i S} + \gamma_i \Delta + \beta(\frac{\gamma_i(\Delta+S)}{1+\gamma_i \Delta})$. However, it is easy to show that also this function

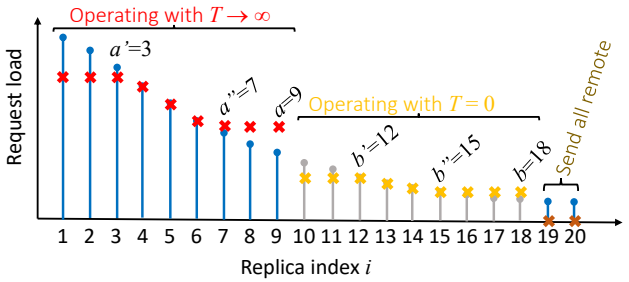


Fig. 4. High-level characteristics of the optimal solution with Dynamic M/M/1 Servers, as defined by Theorems 1-3. **Key insights:** First, the server sites naturally divide into three categories based on their local load: (1) *highly loaded* servers remain always on ($T \to \infty$) and may offload or receive traffic to balance queueing delays, (2) *moderately loaded* servers shut down immediately when idle ($T = 0$) and may engage in load balancing depending on demand, and (3) *lightly loaded* servers do not serve any traffic, forwarding all requests to remote replicas. Second, within the two active groups, the load is further optimized by (1) the *most loaded* sites (in that group) performing load balancing by sending request to others to avoid too high delays, (2) the *least loaded* sites (within that group) serving requests from other sites so to achieve sufficient activity level, and (3) the *intermediately loaded* sites (within that class) serves only the requests of their local clients. Finally, zero or more of the above server site classes may be empty.

is convex with both positive derivative and second derivative. In particular, $\frac{dg}{d\gamma} = \frac{S}{1-\gamma_i S} + \frac{\gamma_i S^2}{(1-\gamma_i S)^2} + \Delta + \beta(\frac{\Delta+S}{1+\gamma_i\Delta} - \frac{\gamma_i\Delta(\Delta+S)}{(1+\gamma_i\Delta)^2}) = \frac{S}{1-\gamma_i S} + \frac{\gamma_i S^2}{(1-\gamma_i S)^2} + \Delta + \beta(\frac{S}{1+\gamma_i\Delta} + \frac{\Delta(1-\gamma_i S)}{(1+\gamma_i\Delta)^2}) > 0$ (since all terms are positive when $\gamma_i S \leq 1$, the range of interest) and $\frac{d^2g}{d\gamma^2} = \frac{2S^2}{(1-\gamma S)^3} + \frac{2\beta\Delta(\Delta+S)}{(1+\gamma\Delta)^3} > 0$. It is therefore always desirable for a partial solution consisting of two replicas with request rates $\gamma + x$ and $\gamma - x$ to be as load-balanced as possible (i.e., small x) and we can apply the same proof as used for the first set also on this set of replicas.

Combined solution: Third, note that since we in both cases have pushed load from left-to-right within each group, we must have $\gamma_a^{new} \geq \gamma_a^{old}$ and $\gamma_{a+1}^{new} \leq \gamma_{a+1}^{old}$. Now, since we by our assumption started with a solution that satisfied Theorem 1, we must have $\gamma_a^{old} \geq \gamma_{a+1}^{old}$. Combining these three observations, we can show that $\gamma_a^{new} \geq \gamma_a^{old} \geq \gamma_{a+1}^{old} \geq \gamma_{a+1}^{new}$. This confirms that also the combined solution satisfies all claimed properties as well as Theorem 1. This completes the proof. □

Combining the insights from Theorems 1-3, we can now define the characteristics of an optimal policy. Figure 4 illustrates such a policy, for an example workload. We note that in this example (as per Theorem 2) the replicas are split into three groups: (1) the replicas with index 1 through a (red markers) operate with $T \to \infty$, (2) the replicas with index $a+1$ through b (yellow markers) operates with $T = 0$, and (3) the least loaded replicas (brown markers) do not serve any requests. Furthermore, for the first two groups (as per Theorem 3) the replicas are split into (up-to) three subgroups: (1) those that serve less than their local demand (which all are load balanced and have the highest local load), (2) those that serve exactly their local load (and have the intermediate loads of the replicas in the group), and (3) those that serve more than only their local load (which are all load balanced and the replicas within the group that have the lowest load). Finally, we note that the load also satisfies the monotonicity property (of non-increasing loads) outlined in Theorem 1.

While our proof outlines a method for converting any solution satisfying the properties outlined in Theorems 1 and 2 into one that also satisfies the properties outlined in Theorem 3, regardless of what starting point is selected (and hence by contradiction that the optimal solution must satisfy the properties claimed in Theorem 3), it is important to note that the method is not guaranteed to find the overall optimal given an arbitrary starting point. First, in practice, the above approach only moves load from left-to-right. If the starting point is perfectly balanced, we may therefore not move any load. Yet, this is often not optimal, and we would have found a better solution with a different starting point. Second, to obtain the overall optimal, we must also consider the relative split of how much load should be placed on the two sets of replicas (i.e., $1 \leq i \leq a$ vs. $a < i \leq b$). With the above approach, we do not consider further improvements that could be obtained by searching for the best split or for the best choices of a and b. We next describe how the above properties can be used to effectively identify the optimal solution.

To find the optimal solution effectively, we use a simple algorithm that first breaks the problem into two optimization problems: (1) find the optimal load split across replicas 1 through a, given a total load A over these replicas, and (2) find the optimal split across replicas $a+1$ through b, given the remaining load (i.e., $\sum_i \lambda_i - A$). Each of these optimizations involve finding the best $\overline{\gamma_a}$ and $\underline{\gamma_a}$ (or $\overline{\gamma_b}$ and $\underline{\gamma_b}$) given a and A (or a, b, and $\sum_i \lambda_i - A$). Both these problems are solved the same way; so let us consider the first. For this function, we note that the allocation is deterministic given A and one of the other parameter; say $\underline{\gamma_a}$. Therefore, given A, we must simply find $\underline{\gamma_a}$. For an effective implementation, we have found it useful here to first loop over all possible replicas a'' ($0 \geq a'' \geq a$), and then for each of them (as long as feasible to find a better solution) we (a) identify feasible bounds of $\underline{\gamma_a}$ (e.g., $\max[\lambda_{a''}, (A - \sum_{i=1}^{a''-1})/(a - a'' + 1)] \leq \underline{\gamma_a} \leq \lambda_{a''-1}$], and (2) search this space for the best a' and $\overline{\gamma_a} = (A - \underline{\gamma_a}(a - a'' + 1) - \sum_{i=a'+1}^{a''-1} \lambda_i)/a')$. Finally, to find the global optimal, we then loop over all possible a, and for each a we search for the best A split and b. (Where for each split we find the best allocation using the above functions.) To speed up the search, we use several monotonicity properties (e.g., $\lambda_1 \geq A/a$, $\frac{1}{N}\sum_i \lambda_i \geq A \geq \sum_i \lambda_i$, $\gamma_a \geq \gamma_{a+1}$, etc.) and prune the searches when possible. Combined, this makes for a relatively efficient search[4] for an optimal solution satisfying the criteria of Theorems 1-3.

D. Optimality with Concave Single-Server Cost

To find the optimal allocation for the *Reactive Unlimited* server system, we first note that the single-server cost for this service class is concave, and then prove a general property that holds for all service classes with a concave cost function.

[4]This implementation was developed for numerical analysis rather than production deployment, so time complexity was not a primary concern. While more efficient algorithms may exist for computing the same optimal solution structure, our current implementation has complexity $\mathcal{O}(N^4 w^2)$, where N is the number of replicas and w is the inverse of the step size used for numerical search. This was sufficient for the resolution and scale of our evaluations.

First, to see the concavity property of the single-server cost of this service class, we define its cost function as:

$$G(\gamma_i) = \gamma_i R(\gamma_i) + \beta C(\gamma_i) = \gamma_i(S+\Delta) + \beta\gamma_i(S + \frac{\Delta}{1+\gamma_i\Delta}), \tag{19}$$

making it is easy to see that $\frac{dG}{d\gamma_i} = (S+\Delta) + \beta(S + \frac{\Delta}{(1+\gamma_i\Delta)^2}) \geq S + \Delta + \beta S$ and $\frac{d^2G}{d\gamma_i^2} = -\frac{2\beta\Delta^2}{(1+\gamma_i\Delta)^3} < 0$.

Second, let us consider the optimal request routing of a set of servers that share a concave cost function $G(\gamma_i)$.

Theorem 4. *When the total cost function can be rewritten as:*

$$\sum_i G(\gamma_i) + \sum_i (\gamma_i - \lambda_i^L) D_R, \tag{20}$$

where $G(\gamma_i)$ is monotonically increasing but with strictly decreasing derivative (i.e., $\frac{dG}{d\gamma_i} \geq 0$ and $\frac{d^2G}{d\gamma_i^2} < 0$), there exists an optimal solution in which the load from the m least loaded replicas are moved to the replica 1. This solution has the following load distribution:

$$\gamma_i = \begin{cases} \lambda_1 + \sum_{i=N-m+1}^{N} \lambda_i, & i = 1, \\ \lambda_i, & 1 < i \leq N-m, \\ 0, & N-m < i \leq N. \end{cases} \tag{21}$$

Proof. We break this proof into three steps. First, we note that any load being moved remotely should be moved to replica 1. This is a direct consequence of $\frac{d^2G}{d\gamma_i^2} < 0$ (and $\frac{dG}{d\gamma_i} \geq 0$) ensuring diminishing additional costs as more load is added to a replica.

Second, it is never beneficial to only partially move the load from a replica. To see this, we note that the above diminishing cost property also ensures that

$$\frac{dG}{dx} G(\gamma_1 + x) + \frac{dG}{dx} G(\lambda_k - x) \leq 0, \tag{22}$$

where γ_1 is the (combined) load on replica 1 before starting to move an additional load x from replica k to replica 1.

Third, we show that if it is beneficial that a replica i moves its load to replica 1, then it is also beneficial for all replicas with index j higher than i. Here, let us consider two example replicas i and j such that $i < j$. Now, since $\frac{d^2G}{d\gamma_i^2} < 0$, we have

$$\frac{d}{dx}\left(\frac{G(\gamma_1 + x) - G(\gamma_1)}{x}\right) \leq 0. \tag{23}$$

For this reason, it is easy to see that:

$$\frac{G(\gamma_1 + \lambda_i + \lambda_j) - G(\gamma_1 + \lambda_i)}{\lambda_j} \leq \frac{G(\gamma_1 + \lambda_i) - G(\gamma_1)}{\lambda_i}, \tag{24}$$

$$\frac{G(\lambda_i)}{\lambda_i} \leq \frac{G(\lambda_j)}{\lambda_j}. \tag{25}$$

Using these two equations, we can now show that the increased cost associated with replica 1 (when moving also the load of replica j to replica 1) increases less than cost change associated with replica j (when moving this load remotely):

$$\begin{aligned} &G(\gamma_1 + \lambda_i + \lambda_j) - G(\gamma_1 + \lambda_i) \leq \frac{\lambda_j}{\lambda_i}(G(\gamma_1 + \lambda_i) - G(\gamma_1)) \\ &\leq \frac{\lambda_j}{\lambda_i}(G(\lambda_i) - \lambda_i R_D) = \frac{\lambda_j}{\lambda_i} G(\lambda_i) - \lambda_j R_D \leq G(\lambda_j) - \lambda_j R_D. \end{aligned} \tag{26}$$

Here, the first inequality uses Equation (24), the second inequality comes from the observation that the load from replica i was moved (hence $G(\gamma_1 + \lambda_i) + \lambda_i R_D \leq G(\gamma_1) + G(\lambda_i)$), and the last inequality comes from Equation (25). This shows that also the load of replica j benefits from being moved.

Combined, the three properties proved above ensure the claimed load distribution of the optimal solution. □

Finally, to find the optimal solution, we must consider all possible such solutions:

$$\min_n \sum_{i=1}^{n} \left(\gamma_i(\Delta + S) + \beta\gamma_i\left(S + \frac{\Delta}{1+\gamma_i\Delta}\right)\right) + D_R \sum_{i=n+1}^{N} \lambda_i, \tag{27}$$

where $\gamma_1 = \lambda_1 + \sum_{i=n+1}^{N} \lambda_i$ and $\gamma_i = \lambda_i$ for $1 < i \leq n$.

E. Optimality with Dynamic M/D/∞ Servers

For this policy class, we define the properties of an optimal solution and show they guarantee optimality. Here, we will use the threshold γ^* defined in Section IV-D, which captures when a server should use $T = 0$ or $T \to \infty$.

Theorem 5. *There exists an optimal multi-replica policy in which (1) any load being moved remotely is moved from the m replicas that have the lowest load and for which the local load $\lambda_i < \gamma^*$, and (2) this load is directed to any of the replicas that always are on (if at least one replica exists with $\gamma^* \leq \lambda_i$) or the replica with the highest load (if no such replica exists). Furthermore, in the case that several replicas have load $\lambda_i \geq \gamma^*$, then it does not matter which of these replicas serves the (extra) load.*

Proof. First, note the following properties of the first and second derivative of Equation (15) for the case when $\gamma_i \geq \gamma^*$:

$$\frac{d}{d\gamma_i}(\gamma_i S + \beta) = S, \qquad \frac{d^2}{d\gamma_i^2}(\gamma_i S + \beta) = 0. \tag{28}$$

Clearly, for the replicas with load beyond this threshold (i.e., $\gamma_i \geq \gamma^*$), it does not matter which replica we give the load.

Second, consider now the case when $\gamma_i < \gamma^*$. For this case, we will repeatedly use that for this case (as derived in Section IV-D) we have that $\gamma_i\Delta(1+\gamma_i\Delta) \leq \beta$. Now, consider first the first derivative of the cost function:

$$\begin{aligned} &\frac{d}{d\gamma_i}(\text{Equation (15)}) = \frac{1}{(\gamma_i\Delta + e^{S\gamma_i})^2}\Big(\Delta^3\gamma_i^2 - \Delta^2 S\gamma_i^2 e^{S\gamma_i} + \Delta^2 S\gamma_i^2 \\ &\quad + 2\Delta^2\gamma_i e^{S\gamma_i} + \Delta e^{S\gamma_i} + \Delta S\gamma_i e^{S\gamma_i} + \beta\Delta + \beta S e^{S\gamma_i} + S e^{2S\gamma_i}\Big) \\ &= S + \frac{1}{(\gamma_i\Delta + e^{S\gamma_i})^2}\Big(\Delta^3\gamma_i^2 - \Delta^2 S\gamma_i^2 e^{S\gamma_i} + \Delta^2 S\gamma_i^2 \\ &\quad + 2\Delta^2\gamma_i e^{S\gamma_i} + \Delta e^{S\gamma_i} - \Delta S\gamma_i e^{S\gamma_i} + \beta\Delta + \beta S e^{S\gamma_i}\Big) \\ &\geq S + \frac{\Delta^3\gamma_i^2 + 2\Delta^2\gamma_i e^{S\gamma_i} + \Delta e^{S\gamma_i} + \beta\Delta}{(\gamma_i\Delta + e^{S\gamma_i})^2} \geq S. \end{aligned} \tag{29}$$

Here, we used that $\gamma_i\Delta(1+\gamma_i\Delta) \leq \beta$ in the second-last step.

We next prove that the second derivative is non-positive (for this case) in three steps.

$$\frac{d^2}{d\gamma_i^2}(\text{Equation}(15)) = -\frac{1}{(\Delta\gamma_i + e^{S\gamma_i})^3} \times$$
$$\times \Big(2\beta\Delta^2 - \beta\Delta S^2\gamma_i e^{S\gamma_i} + 4\beta\Delta S e^{S\gamma_i} + \beta S^2 e^{2S\gamma_i} + \Delta^3 S^2\gamma_i^3 e^{S\gamma_i}$$
$$+ \Delta^2 S^2\gamma_i^2 e^{S\gamma_i} - \Delta^2 S^2\gamma_i^2 e^{2S\gamma_i} - 2\Delta^2 S\gamma_i e^{S\gamma_i} + 4\Delta^2 S\gamma_i e^{2S\gamma_i}$$
$$+ 2\Delta^2 e^{S\gamma_i} - 2\Delta^2 e^{2S\gamma_i} - \Delta S^2\gamma_i e^{2S\gamma_i} + 2\Delta S e^{2S\gamma_i} \Big) \quad (30)$$

To simplify, let us break out Δ^2 and introduce two substitute variables: $y = \gamma_i\Delta$ and $z = \gamma_i S$. We then have the expression:

$$\frac{d^2}{d\gamma_i^2}() = -\frac{\Delta^2}{(y+e^z)^3} \times \Big(2\beta - \beta\frac{z^2}{y}e^z + 4\beta\frac{z}{y}e^z + \beta\frac{z^2}{y^2}e^{2z} + yz^2e^z$$
$$+ z^2e^z - z^2e^{2z} - 2ze^z + 4ze^{2z} + 2e^z - 2e^{2z} - \frac{z^2}{y}e^{2z} + 2\frac{z}{y}e^{2z} \Big) \quad (31)$$

To show that this expression is non-positive, we must show that the expression within the bracket is non-negative. To do this, we first show that the sum of the four initial terms (all with a factor β) always is non-negative. Ignoring all other terms, gives $2\beta - \beta\frac{z^2}{y}e^z + 4\beta\frac{z}{y}e^z + \beta\frac{z^2}{y^2}e^{2z}$. After multiplying by y^2 and solving for where this expression is zero:

$$2y^2 - y(z^2e^z + 4ze^z) + z^2e^{2z} = 0, \quad (32)$$

we note that the two only roots

$$y_{1,2} = \frac{1}{4}\Big((z-4)ze^z \pm ze^z\sqrt{z^2-8z+8} \Big), \quad (33)$$

either are imaginary or negative when $z > 0$. This shows that the sum of these four terms always is non-negative. For this reason, we know that the sum of these terms always is no smaller than for the case when $\beta = y(y+1)$ (i.e., $\beta = \gamma_i\Delta(\gamma_i\Delta+1)$) and $\frac{d}{d\beta}$(Equation (31)) is non-positive. We can therefore use $\beta = y(y+1)$ as a conservative bound in Equation (31). After insertion and collecting terms, we then have that:

$$\frac{d^2}{d\gamma_i^2}(\text{Equation}(15)) \Big/ \frac{\Delta^2}{(y+e^z)^3} \leq$$
$$\leq -\Big(2y(y+1) - (y+1)z^2e^z + 4(y+1)ze^z + (y+1)\frac{z^2}{y}e^{2z} + yz^2e^z$$
$$+ z^2e^z - z^2e^{2z} - 2ze^z + 4ze^{2z} + 2e^z - 2e^{2z} - \frac{z^2}{y}e^{2z} + 2\frac{z}{y}e^{2z} \Big)$$
$$\leq -\Big(2y^4 + y^3(2+4ze^z) + y^2(2ze^z + 4ze^{2z} + 2e^z - 2e^{2z}) + y(2ze^{2z}) \Big)$$
$$\leq -\Big(2y^4 + y^3(2+4ze^z) + y^2(2ze^z + 2ze^{2z}) + y(2ze^{2z}) \Big) \leq 0 \quad (34)$$

Here, in the second-last step, we used that $2ze^{2z} + 2e^z - 2e^{2z} \geq \max[2ze^{2z}, 2e^z] - 2e^{2z} \geq 0$, noting that $ze^{2z} \geq e^{2z}$ whenever $z \geq 1$ and $e^z \geq e^{2z}$ whenever $0 \leq z \leq 1$. Finally, in the last step we note that all terms within the brackets are non-negative (when $y \geq 0$ and $z \geq 0$) and the first factor is negative, resulting in a non-positive product. With a positive derivative and non-positive second derivative, the rest of the proof follows the same logic as the proof for a single concave cost function, completing the proof. □

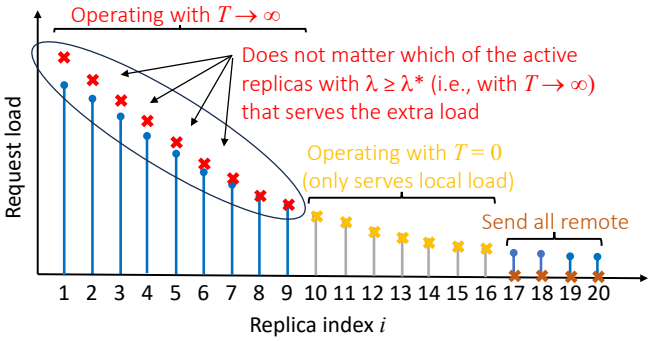


Fig. 5. High-level characteristics of the optimal solution with Dynamic M/D/∞ Servers, as defined by Theorem 5. **Key insights:** First, the server sites naturally divide into three categories based on their local load: (1) *highly loaded* servers remain always on ($T \to \infty$) and may receive traffic from other sites, (2) *moderately loaded* servers shut down immediately when idle ($T = 0$) only server local requests, and (3) *lightly loaded* servers do not serve any traffic and forward all their requests to remote replicas.

Figure 5 shows a high-level characterization of the optimal solution with *Dynamic M/D/*∞ servers, as defined by Theorem 5. In this example, the load originally associated with replicas 17-20 (i.e., the brown replicas, which have the lowest local demand) are direct to one or more of replicas 1-9, which all have local request rate above λ^* and operate with $T \to \infty$. Finally, the replicas with intermediate load (i.e., replicas 10-16) only serve their local requests and operate with $T = 0$.

Calculating optimal cost: While the specific choice of replica(s) in the first category is arbitrary, when no such replica exists, it is optimal to allocate load to the most heavily loaded site. Therefore, the simplest policy is to always redistribute the load of the m least loaded replica sites to the most loaded one (i.e., exactly the optimal strategy under concave cost functions, as discussed in Section V-D). With this policy, we simply need to consider one of $N_{\lambda_i<\lambda^*}$ possible configurations, where $\min[N-1, N_{\lambda_i<\lambda^*}]$ is the number of replicas with local load below the threshold λ^*:

$$\min_{0\leq m\leq \min[N-1, N_{\lambda_i<\lambda^*}]} \sum_{i=1}^{N-m} \text{cost}(\gamma_i), \quad (35)$$

where cost(γ_i) is given by Equation (15), with $\gamma_1 = \lambda_1 + \sum_{i=N+1-m}^{N}\lambda_i$, $\gamma_i = \lambda_i$ for $2 \leq i \leq N-m$, and $\gamma_i = 0$ otherwise.

VI. Numerical Comparisons with Baseline Policies

A. Baseline Policies

Simpler policies are typically implemented and used in practice. Using the optimal policies derived in previous sections, we next provide insights into the relative performance of such policies. Specifically, we compare three baseline policies with the optimal policy associated with each service class. The policies of consideration are the following.

- **Local:** All requests are directed and served by the closest replica server. In this case, the total cost reduces to:

$$\sum_i \lambda_i R(\lambda_i) + \beta\sum_i C(\lambda_i). \quad (36)$$

- **Balanced:** The requests are perfectly load balanced across the replica sites, while (under that condition) as

many requests are served locally as possible. In this case, the total cost expression can be written:

$$\sum_c \max[0, \gamma^* - \lambda_c] D'_R + |\mathcal{N}|(\gamma^* R(\gamma^*) + \beta C(\gamma^*)), \tag{37}$$

where $\gamma^* = \frac{1}{|\mathcal{N}|} \sum_i \lambda_i$

- **Single:** All requests are directed and served by the most loaded replica site. In this case, the total cost reduces to:

$$(\gamma^{tot} - \lambda_1) D'_R + \gamma^{tot} R(\gamma^{tot}) + \beta C(\gamma^{tot}), \tag{38}$$

where $\gamma^{tot} = \sum_i \lambda_i$.

To calculate these costs for each service class, we simply use the response time function $R(\gamma)$ and cloud-service cost $C(\gamma)$ associated with the service class of interest.

While simple, each of these baseline policies reflects a distinct and practically relevant routing strategy. The *Local* policy is well-suited when remote access delays are significant, as it minimizes network transfer costs and is often favored in latency-sensitive applications. The *Balanced* policy performs best when remote costs are low and minimizing queueing delay is critical, such as under high utilization or when services are delay-sensitive (e.g. want to have low utilization under *Dynamic M/M/1* scenarios). The *Single* policy, though initially unintuitive in a distributed setting, can be effective when the cost per request decreases with scale, as in service classes like *Dynamic M/D/*∞ or *Reactive Unlimited*, where parallelism or quick scaling reduces delays, especially when the remote delay cost is small. Including this policy highlights how a strategy that is optimal in one context (e.g., high parallelism) can be substantially suboptimal in others (e.g., queueing-based M/M/1), reinforcing the need for class-specific optimization. We also note that demand for different services delivered by the same infrastructure may see different demands.

B. When Do Baseline Policies Perform Well or Poorly?

To understand when each of the above baselines are more or less useful, we compared their relative performance for a wide range of workload and system parameters. Figures 6-9 show their normalized relative increase in cost (as a percentage) compared to the optimal solution for each service class. Throughout these evaluations, again use normalized units, with the average service time set to $S = 1$, so that costs and delays are expressed in units of service time. This ensures consistency and generality across comparisons. For these results, we have selected a default scenario ($N = 16$, $\lambda = 0.2$, $\Delta = 1$, $D_R = 1$, and $\beta = 1$) and then vary one parameter at a time one magnitude in each direction. We next discuss these one-factor results one service class at a time.

Individual service case: For the case when each request is individually served, it is always optimal to serve all requests locally and there are no benefits to direct extra requests to a particular server. Furthermore, with more load being shifted with the *single* policy than the *balancing* policy, for the scenarios considered here, the balanced policy consistently outperforms sending all requests to a single replica. We also note that the penalties of these sub-optimal policies are biggest when the remote access delays (D_R) are big. Shorter startup times (Δ) and smaller cloud cost weights (β) also increase the performance differences between the policies.

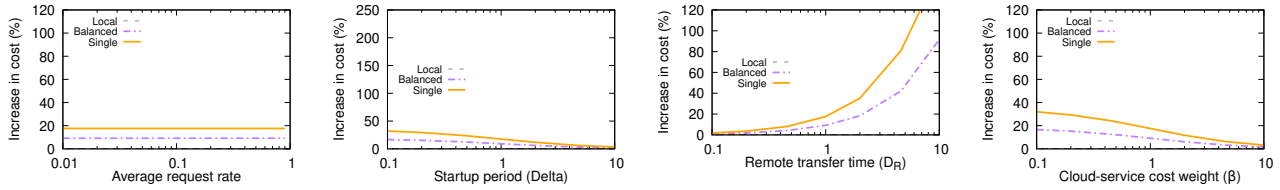
Dynamic M/M/1 server case: For the M/M/1 service case, the single replica policy is only feasible for small arrival rates. While the local policy extends the region of reasonable costs somewhat, with the exception of low request rates (where it is optimal), it is typically outperformed by the balanced policy. However, note that also here there are significant regions of the parameter space, where also this policy is substantially sub-optimal (e.g., when startup or remote transfer times grow large). These results both highlight the value in carefully optimizing how requests are distributed across the replicas and the fact that the best policy for one service class (e.g., local service with individual service) sometimes can be highly sub-optimal for a different service class (e.g., *Dynamic M/M/1*).

Reactive Unlimited server case: For this case, the local policy again performs well: typically within 20% of optimal (except for large β). However, there are substantial regions where one of the other policies is better (and even optimal). For example, while local is optimal for small startup delays (e.g., $\Delta \leq 1$), large remote transfer costs (e.g., $D_R \geq 1$), and small cloud-service costs (e.g., $\beta \leq 1$), the single server policy is optimal for large startup periods (e.g., $\Delta \geq 2$), small remote transfer times (e.g., $D_R \leq 0.5$), and high cloud-service cost weights (e.g., $\beta \geq 2$). Furthermore, in contrast to the M/M/1 case, there is no region where the balanced policy is optimal or even the best of the baselines.

Dynamic M/D/∞ server case: Finally, for the M/D/∞ case, we have found that the single server case often is optimal or close to optimal for the parameter ranges considered here. However, we observe regions where it is sub-optimal and in some cases is substantially outperformed by one or both of the other baseline policies; e.g., when the remote transfer costs grow big or when the cloud-service cost weights are small. For these cases, it is instead typically better to use the local policy. Like for the unlimited server case, the balanced policy is always outperformed by the local policy. For these last two service cases, we note that close to optimal performance can instead be achieved by always taking the better of either always sending all requests to a single replica or serving all requests locally, whichever is the better of the two. In the M/D/∞ case, this hybrid heuristic achieves within 10% of optimal throughout the parameter ranges considered here.

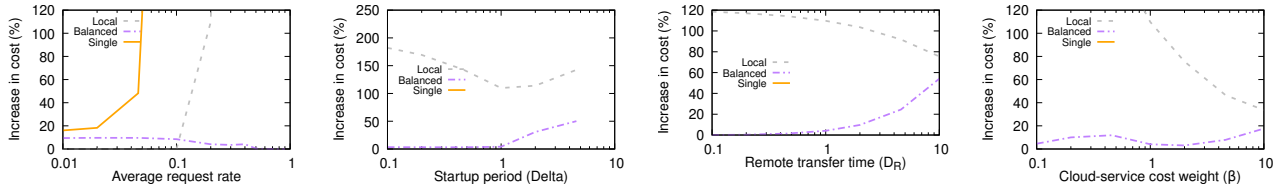
C. Comparisons Over the Service Classes

The comparative analysis across service classes reveals nuanced insights into the performance of load-balancing policies under varying workload and system conditions. While the local policy typically outperforms the balanced policy, except for the M/M/1 service case, it is sometimes outperformed by the single replica policy in the the unlimited server or M/D/∞ cases, where a single site can handle high request rates efficiently. The efficacy of other policies varies even more depending on the service class and operational context. These findings underscore the importance of tailoring load-balancing strategies to specific service requirements and system characteristics, while also highlighting the potential for



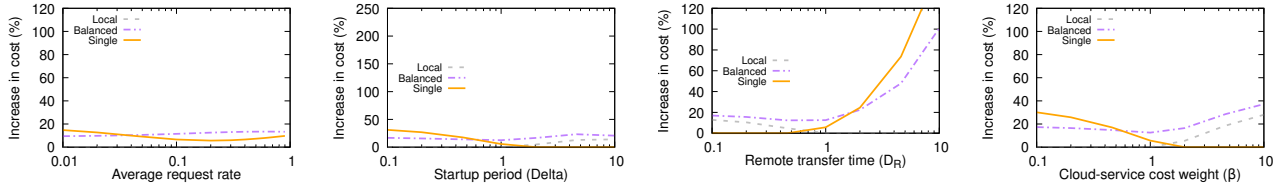
(a) Average request rate (λ) (b) Startup times (Δ) (c) Remote access delay (D_R) (d) Cloud-service cost weights (β)

Fig. 6. Baseline comparisons when using Individual service. Costs are expressed in normalized time units, assuming $S = 1$, and shown as percentage (%) increases relative to the optimal policy for this class. Default values are: $N = 16$, $\alpha = 1$, $\lambda = 0.2$, $\Delta = 1$, $D_R = 1$, and $\beta = 1$. **Key observation:** *Local* service is optimal for this service class.



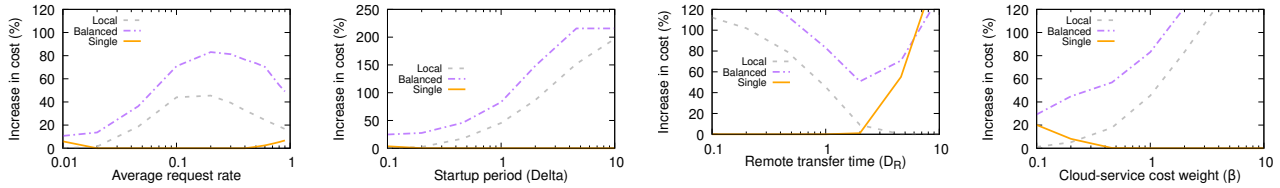
(a) Average request request rate (λ) (b) Startup times (Δ) (c) Remote access delay (D_R) (d) Cloud-service cost weights (β)

Fig. 7. Baseline comparisons when using Dynamic M/M/1. Costs are expressed in normalized time units, assuming $S = 1$, and shown as percentage (%) increases relative to the optimal policy for this class. Default values are: $N = 16$, $\alpha = 1$, $\lambda = 0.2$, $\Delta = 1$, $D_R = 1$, and $\beta = 1$. **Key observation:** While seldom optimal, for this service class *Balanced* typically perform best of the simple policies.



(a) Average request rate (λ) (b) Startup times (Δ) (c) Remote access delay (D_R) (d) Cloud-service cost weights (β)

Fig. 8. Baseline comparisons when using Reactive Unlimited Server system. Costs are expressed in normalized time units, assuming $S = 1$, and shown as percentage (%) increases relative to the optimal policy for this class. Default values are: $N = 16$, $\alpha = 1$, $\lambda = 0.2$, $\Delta = 1$, $D_R = 1$, and $\beta = 1$. **Key observation:** For this secrvice class, we do not observe any case when *Balanced* is optimal or even the best choice of the simple policies. Instead, *Local* service often perform well (but not optimally) for large portion of the parameter space.



(a) Average request rate (λ) (b) Startup times (Δ) (c) Remote access delay (D_R) (d) Cloud-service cost weights (β)

Fig. 9. Baseline comparisons when using Dynamic M/D/∞. Costs are expressed in normalized time units, assuming $S = 1$, and shown as percentage (%) increases relative to the optimal policy for this class. Default values are: $N = 16$, $\alpha = 1$, $\lambda = 0.2$, $\Delta = 1$, $D_R = 1$, and $\beta = 1$. **Key observation:** For this service class, while *Single* server sometimes is significantly outperformed even by the other simple policies, this policy performs optimally for large portions of the considered parameter space. This captures the benefits of service aggregation in the M/D/∞ case.

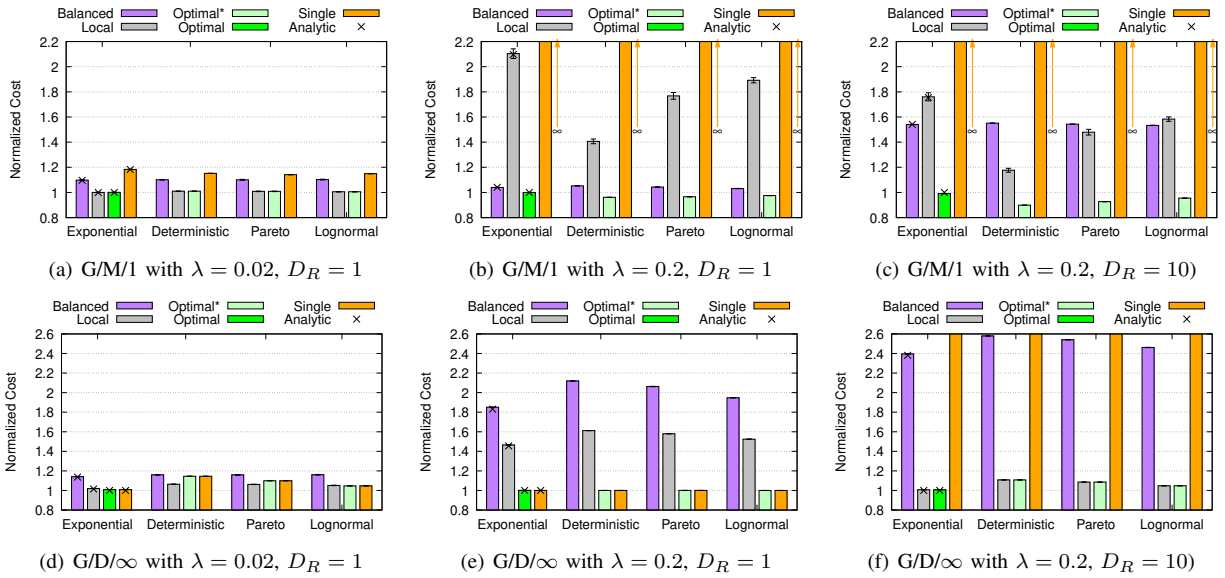
hybrid approaches to offer robust performance across diverse conditions. For example, in all cases except the M/M/1 service case, a hybrid of local and single replica would achieve close to optimal (e.g., for the M/D/∞ service class within 10% of optimal for the parameter ranges considered). In contrast, for the M/M/1 service case, the balanced policy typically is the best of the baseline policies, capturing the importance of keeping the response times under check.

D. Sensitivity Analysis of Distribution Assumptions

We next explore how sensitive the observed performance is to the specific distribution assumptions made in the analytical models. Specifically, we examine the impact of using alternative distributions for inter-arrival times, service times, startup times, and shutdown times across two most extreme service class representatives: the *Dynamic M/M/1* and *Dynamic M/D/*∞ models.

Focusing on relative request routing policy comparisons, for these simulations, we use the request routing policies to assign request rates to each replica site, simulate each replica site independently, and then aggregate the results across sites. This modeling approach fully and accurately captures the impact of different service time, startup time, and shutdown time distributions, provided that arrivals follow a Poisson process.

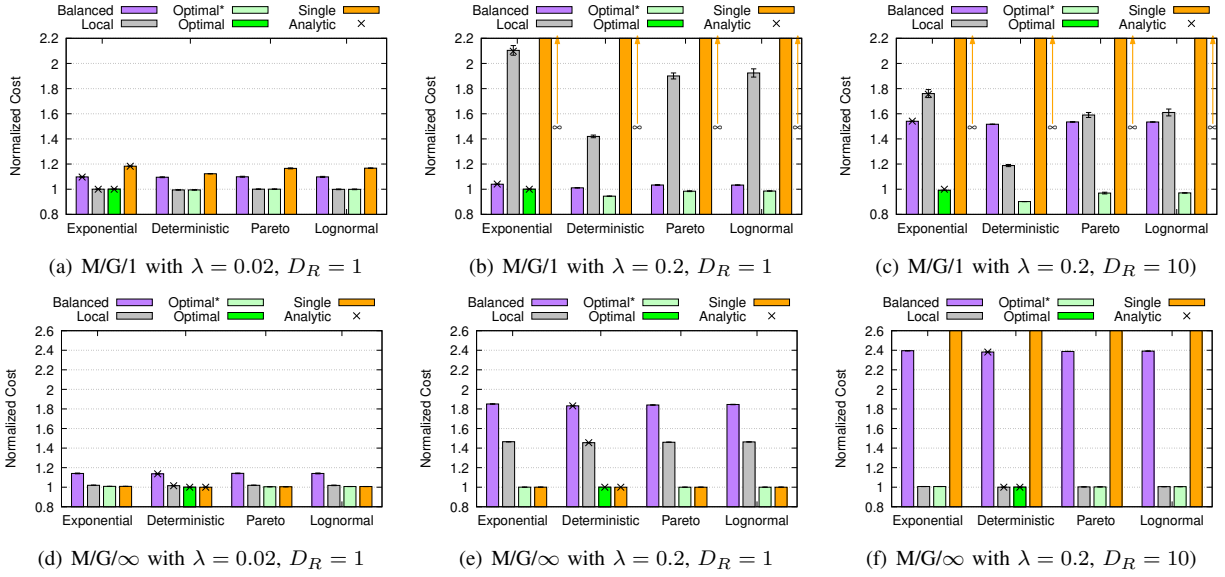
In the following we include results for generalizations of all four of these distributions, noting that the simulations are accurate for the first three generalizations but can only be seen as an approximation for non-Poisson arrivals. For non-Poisson arrivals, while the model does not explicitly capture instantaneous state correlations between replica sites, it still



(a) G/M/1 with $\lambda = 0.02$, $D_R = 1$ (b) G/M/1 with $\lambda = 0.2$, $D_R = 1$ (c) G/M/1 with $\lambda = 0.2$, $D_R = 10$)

(d) G/D/∞ with $\lambda = 0.02$, $D_R = 1$ (e) G/D/∞ with $\lambda = 0.2$, $D_R = 1$ (f) G/D/∞ with $\lambda = 0.2$, $D_R = 10$)

Fig. 10. Impact of request inter-arrival patterns, emulated here by one of four distributions: Exponential, Deterministic, Pareto, and Lognormal. Here, we normalize all values with the total cost of the optimal solution under Poisson arrivals (i.e., exponential distribution). Default values are: $N = 16$, $\alpha = 1$, $S = 1$, $\Delta = 1$, $T = 1$, and $\beta = 1$. (Note that the "optimal" request routing policy is only guaranteed to be optimal under our modeling assumptions.)



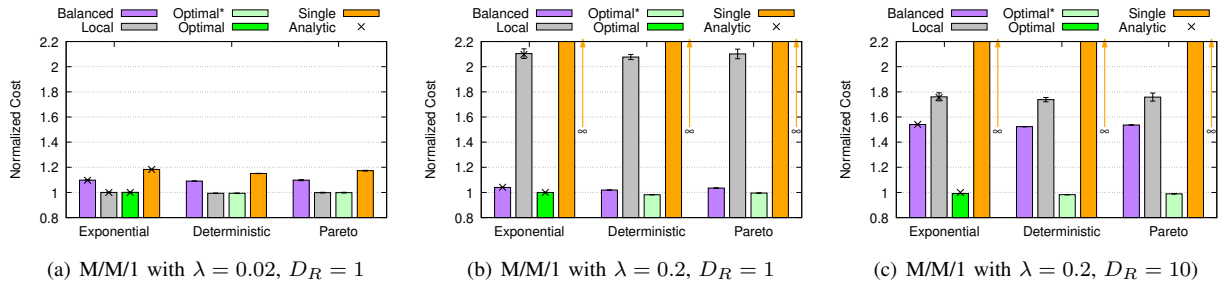
(a) M/G/1 with $\lambda = 0.02$, $D_R = 1$ (b) M/G/1 with $\lambda = 0.2$, $D_R = 1$ (c) M/G/1 with $\lambda = 0.2$, $D_R = 10$)

(d) M/G/∞ with $\lambda = 0.02$, $D_R = 1$ (e) M/G/∞ with $\lambda = 0.2$, $D_R = 1$ (f) M/G/∞ with $\lambda = 0.2$, $D_R = 10$)

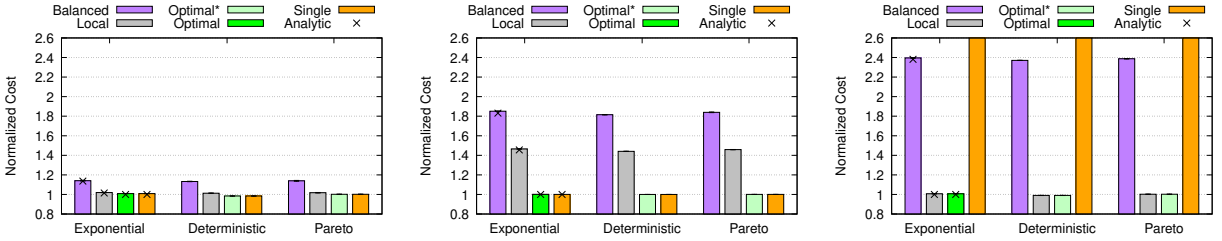
Fig. 11. Impact of service time distribution, here exemplified by one of four distributions: Exponential, Deterministic, Pareto, and Lognormal. Similar to above, we again normalize all values with the total cost of the optimal solution under our default distributions: exponential (M/M/1) and deterministic (M/D/∞) distributions, respectively. Default values are: $N = 16$, $\alpha = 1$, $S = 1$, $\Delta = 1$, $T = 1$, and $\beta = 1$. (Note that the "optimal" request routing policy is only guaranteed to be optimal under our modeling assumptions.)

retains the essential dynamics of how different policies shape load distribution and system behavior. As such, the approach provides a reasonable and informative approximation that enables meaningful comparison across policy classes under diverse arrival patterns. Furthermore, given that our primary focus is to evaluate *relative policy performance* rather than to develop precise request splitting heuristics under state-dependent arrival processes, we believe that including also the generalized arrival time distribution results offers meaningful insight. In practice, many systems employ simple probabilistic routing under such conditions, making our approximations both relevant and useful for capturing key trends in policy behavior across a broad range of workloads.

Figures 10-13 present simulation results for both service classes (G/G/1 and G/G/∞) under various distributional settings. For each figure, we consider three different traffic scenarios (labeled (a), (b), and (c) for the M/M/1 case and (d), (e), and (f) for the M/D/∞ case). For each scenario, the performance of all four load balancing policies (Local, Balanced, Single, and Optimal) are shown, along with 95% confidence intervals based on 10 independent simulation runs. To allow direct comparison to our earlier analytic results, each figure also includes the corresponding analytic values for the default M/M/1 (or M/D/∞) with exponential or deterministic distributions (including for the default exponential startup times Δ and the optimized shutdown period T). Furthermore, all cost values are normalized with regards to the cost of the optimal policy (under our original model assumptions),

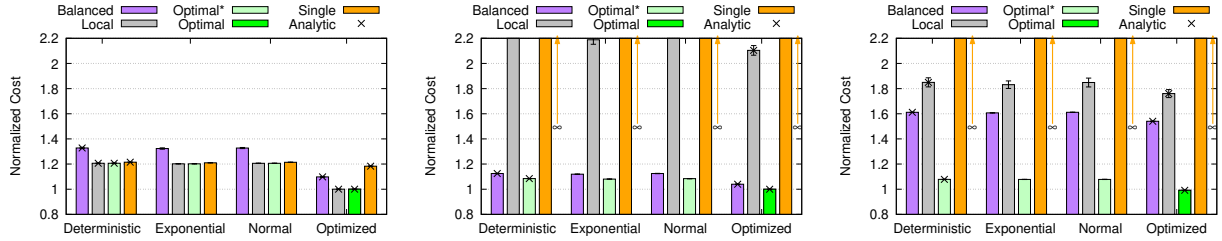


(a) M/M/1 with $\lambda = 0.02$, $D_R = 1$ (b) M/M/1 with $\lambda = 0.2$, $D_R = 1$ (c) M/M/1 with $\lambda = 0.2$, $D_R = 10$

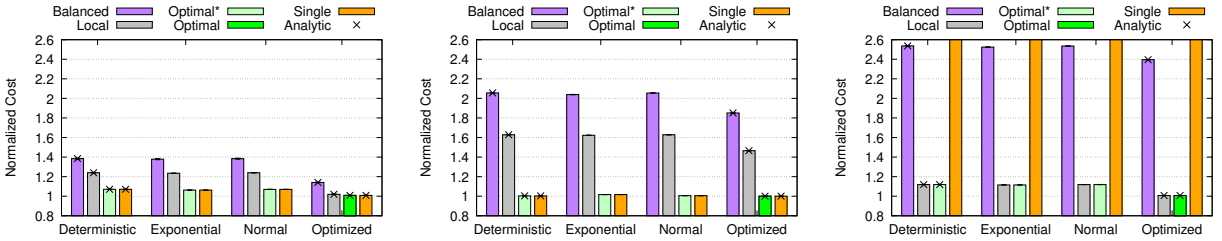


(d) M/D/∞ with $\lambda = 0.02$, $D_R = 1$ (e) M/D/∞ with $\lambda = 0.2$, $D_R = 1$ (f) M/D/∞ with $\lambda = 0.2$, $D_R = 10$

Fig. 12. Impact of startup-time time distribution, here exemplified by one of three distributions: Exponential (default), Deterministic, and Pareto. Similar to above, we again normalize all values with the total cost of the optimal solution under our default distributions: exponential Δ. Default values are: $N = 16$, $\alpha = 1$, $S = 1$, $\Delta = 1$, $T = 1$, and $\beta = 1$. (Note that the "optimal" request routing policy is only guaranteed to be optimal under our modeling assumptions.)



(a) M/M/1 with $\lambda = 0.02$, $D_R = 1$ (b) M/M/1 with $\lambda = 0.2$, $D_R = 1$ (c) M/M/1 with $\lambda = 0.2$, $D_R = 10$



(d) M/D/∞ with $\lambda = 0.02$, $D_R = 1$ (e) M/D/∞ with $\lambda = 0.2$, $D_R = 1$ (f) M/D/∞ with $\lambda = 0.2$, $D_R = 10$

Fig. 13. Impact of shutdown-time distribution, here exemplified by one of four distributions: Deterministic (default), Exponential, and Normal. Similar to above, we again normalize all values with the total cost of the optimal solution under our default distribution: deterministic T. Default values are: $N = 16$, $\alpha = 1$, $S = 1$, $\Delta = 1$, $T = 1$, and $\beta = 1$. (Note that the "optimal" request routing policy is only guaranteed to be optimal under our modeling assumptions.)

allowing us to not only compare both the impact that different distributions have on the overall cost and on the relative performance of the different request routing policies.

For each scenario, we evaluated the effect of using four different inter-arrival distributions (exponential, deterministic, Pareto, and lognormal), four different service time distributions (same four types), three startup time distributions (exponential, deterministic, and Pareto), and three shutdown time distributions (deterministic, exponential, and normal). These distributions cover an extremely wide span of system behaviors, ranging from highly regular to highly variable and heavy-tailed patterns. Importantly, in all cases we adjust parameters so that the average rate or time remains the same across distributions, thus isolating the impact of distributional shape rather than mean differences.

First, and most importantly, we have found that the relative performance of the policies often are similar for the other distributions than those of the original models. Here, it is important to note that the "optimal" policy is based on the analytically derived request allocation (using our modeling assumptions for the M/M/1 and M/D/$infty$ cases) and should not be expected to correspond to the best request routing allocation also for other distribution assumptions. It is therefore impressive to see that this optimized request routing policy (with optimizations done over the original modeling assumptions) typically is the best performing policy. Combined, these results suggest that the optimized request routing based on our models can be effective in more complex scenarios and that

the relative performance comparisons and insights obtained under the model are generalizable.

Overall, while there are some exceptions and case-specific variations, the results across Figures 10-13 suggest that the choice of distribution does not substantially alter the relative performance ranking between the policies. In particular, for both the M/M/1 and M/D/∞ service models, we observe that the primary trends and policy comparisons remain relatively consistent across distribution types. This robustness holds across all three tested workload scenarios.

For example, even with heavy-tailed (Pareto) or high-variance (lognormal) distributions, the relative benefits of the optimal policy compared to the baseline policies remain largely predictable and in line with the analytic results derived under the exponential assumptions. While some performance degradation can be observed in certain scenarios (particularly for the single policy under heavy-tailed service times), the overall impact of the distributional choice on total system cost and policy selection appears limited.

In summary, these experiments suggest that while workload and system variability can have some effect on absolute performance, the relative performance gaps between policies remain relatively stable. This robustness further strengthens the generalizability of our findings, suggesting that key conclusions drawn for the default distributions and use case scenarios also extend to more diverse and realistic workload distributions.

E. Discussion: Additional Constraints and Priorities

Our model jointly optimizes response delay and server usage cost under a unified cost function, capturing key system tradeoffs in a tractable form. While this abstraction enables rigorous analysis and policy derivation, we acknowledge that real-world systems often operate under additional constraints or heterogeneous requirements. For example, some services impose strict upper bounds on acceptable response times, while others prioritize specific request classes over others based on business or user importance.

Although our framework does not explicitly model such constraints, it provides a foundational structure for understanding the impact of service class design and routing strategies. Extensions to our model could incorporate heterogeneous delay penalties across request types by introducing class-specific cost weights. For some service models (e.g., *Dynamic M/M/1* or *Reactive Unlimited*), this would also raise scheduling considerations across request classes. While first-come-first-served (FCFS) scheduling retains analytical tractability under exponential assumptions, more general service-time distributions or non-FCFS disciplines would lead to significantly more complex models. Similarly, in the *Dynamic M/D/*∞ case, one practical extension might be to provision one server per request type, avoiding contention altogether.

Analytic treatment of such extended models is challenging but may be possible in certain cases (e.g., using delay distribution expressions in M/G/1-type models or applying simulation). We leave such richer models for future work. The goal of this paper is to develop core insights under clean, analyzable models that isolate the key performance-cost tradeoffs associated with different service class designs.

Finally, we note that our evaluation already considers general arrival-rate and service-time distributions (G/G/1, G/G/∞) suggest that the key tradeoffs our model captures remain relevant even under more complex system dynamics.

F. Practical Implications and Real-World Relevance

The results in this paper, while presented using normalized units, reflect tradeoffs that are directly relevant in real-world cloud deployments. For instance, in serverless or autoscaling systems (e.g., AWS Lambda, Kubernetes, or Google Cloud Run), both startup latency and server usage costs are central concerns. Startup delays may range from hundreds of milliseconds to several seconds due to container warm-up, while server costs are often billed per second or per GB-hour.

Our results demonstrate that simple routing policies (e.g., always using the local server or evenly balancing load) can lead to significantly higher cost or delay, depending on system load and service class characteristics. As shown in Figures 6–9, the cost gap between optimized and baseline policies can be substantial, particularly when server startup delays are high or when remote transfer times are non-negligible. For example, a system incurring 1–2 seconds of cold-start latency could, under high load, suffer significantly performance or cost penalties without the insights offered by our optimal policies.

Thus, even without specific deployment parameters, our findings illustrate that commonly deployed heuristics (e.g., local-first or uniform balancing) can be substantially suboptimal. This highlights the importance of matching request routing strategies to the underlying service class, rather than relying on generic heuristics; a key contribution of this work.

VII. Related Work

Dynamic allocation: Several works have developed or applied analytic models for systems with server setup delays. For example, Meisner et al. [11] applied an early analysis of an M/G/1 single server system [12] for energy conservation through rapid transitions between active and near-zero-power states. Subsequent studies have explored multi-server systems with exponentially distributed setup delays [13], [14], [15], [16], considered deterministic setup delays [17], or have focused on systems with delayed-off policies, where servers are deallocated after a delay following idle periods [18], [19], [15]. Other works have considered dynamic server [20] and content [21] allocation for the purpose of cloud-operated caches. Most closely related to this work is the recent work by Carlsson and Eager [1], which derives the first closed-form expressions for the single-site M/M/1 and unlimited server systems. In the present paper, we use those results (e.g., Equations (6) and (10)) as components in a broader analysis of optimized request routing across multiple replicas and service classes. Unlike [1], which focuses on single-site server allocation, this paper addresses the joint routing and cost optimization problem in multi-site systems, compares four service models, and derives structural insights into optimal request routing across dynamically provisioned resources. None of the above works considered the combined problem of server allocation/deallocation and request routing.

Request routing: While some works have considered routing for geographically distributed cloud services [22], most prior work has focused on server and replica placement in content delivery networks (CDNs). Here, iterative greedy heuristics have proven effective for minimizing network costs [23], [24], [25], [26], while others have optimized delay relative to server placement [27], improved client-to-server mapping via IP prefix [28], or adapted the number of replicas based on demand [29]. Additional studies have addressed cooperative caching and request routing [30], efficient content distribution to replicas [31], [32], [33], [34], and joint CDN-ISP infrastructure reconfiguration [35].

More recent works have considered joint service placement and request routing in MEC and edge environments [36], [37], [38], [39], often under multidimensional constraints (e.g., storage, computation, communication). However, these models typically assume statically provisioned or long-lived server allocations and rely on approximate optimization methods (e.g., MILPs, randomized rounding). In contrast, we focus on request routing in systems where server instances are provisioned dynamically on demand, with both cost and delay driven by runtime activity—capturing behaviors such as startup overheads and elastic scaling.

This dynamic setting changes the structure of the routing problem: decisions not only determine where requests are served, but also when servers are instantiated, how long they stay active, and the resulting resource costs. Our framework enables closed-form analysis and structural insights into optimal routing policies across multiple service classes, without relying on heuristics or NP-hard formulations. Unlike prior work, we explicitly model server life cycles and usage-driven cost, which are central to modern cloud platforms with autoscaling and serverless capabilities.

In addition to request routing where each request is served individually [28], [36], [35], [40], [41], [42], [43], some works have considered the problem in the context of aggregated services [44], [45], [46], [47], [48] or considered the combined problem of optimized cache copy placement and request routing using TTL-based [49], [50], [51] or LRU-based [52], [53] caches. While some of these models could be loosely mapped to our *individual service* or *M/D/*∞ service classes, the core difference is that these prior works assume either fixed server resources or static replica placement. In contrast, our work explicitly captures the dynamic life cycle of server instances, including startup overhead, cost accumulation only during active usage, and request-driven activation and deactivation. To our knowledge, no prior work has integrated these dynamic behaviors into the joint routing and cost optimization problem.

In summary, prior work on request routing and server placement typically assumes statically provisioned resources, where the cost of a server is independent of load or activation frequency. In contrast, our work considers systems with on-demand server instantiation, usage-based cost, and startup overheads, which fundamentally alter the optimality conditions for routing. By modeling these dynamic behaviors explicitly, we derive structural properties of optimal routing policies across four service classes—ranging from serialized to highly parallel systems. Rather than relying on heuristics or approximations for NP-hard formulations, our approach enables closed-form analysis and highlights new decision boundaries that are essential in modern elastic cloud environments such as serverless and autoscaling platforms.

VIII. Conclusions

In this paper, we have provided an in-depth analysis and evaluation of cost-optimized request routing policies for cloud-based systems using different service classes. Through the development of a comprehensive system model, derivation of an exact analysis, derivation of optimized policies, and careful evaluation and comparison against basic baseline policies, we shed light into the intricate interplay between resource allocation, request routing, and system performance, and offer valuable insights into system optimization for these systems.

Our research makes several key contributions. First, we present optimized request routing policies tailored to different service classes, derived through exact single-site analyses and extended to multi-replica routing solutions. Second, we conduct a comprehensive evaluation and comparison of these policies, highlighting their relative performance and unveiling scenarios where certain strategies excel or falter. Notably, our examination of optimal solutions provides a benchmark for assessing the performance of simpler, more practical policies. Finally, and most importantly, our findings offer insights into the optimization of cloud-based systems, emphasizing the significance of workload characteristics, system parameters, and load-balancing strategies. By comparing optimal solutions across different service classes, we showcase when basic policies may suffice, when advanced strategies are warranted, and the potential efficacy of hybrid approaches that amalgamate the strengths of multiple policies.

In summary, our research advances the understanding and refinement of load-balancing techniques in cloud-based systems, paving the way for enhanced performance and efficiency in distributed computing environments. As cloud computing continues to evolve, our work provides a foundation for future endeavors aimed at optimizing resource utilization and maximizing system throughput.

Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research (SSF), project FUS21-0033, and the Swedish Research Council (VR), grant 2022-04690.

References

[1] N. Carlsson and D. Eager, "Quantifying the performance gap for simple versus optimal dynamic server allocation policies," *arXiv preprint arXiv:2507.19667*, 2025, submitted, May 2024.

[2] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX ATC*, 2020.

[3] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *Proc. NSDI*, 2020.

[4] A. Joosen, A. Hassan, M. Asenov, R. Singh, L. Darlow, J. Wang, and A. Barker, "How does it function? characterizing long-term trends in production serverless workloads," in *Proc. ACM SoCC*, 2023.

[5] Y. Zhang, Íñigo Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *Proc. ACM SOSP*, 2021.

[6] "Google cloud run," https://cloud.google.com/run, accessed: July 2025.

[7] "Ibm code engine," https://www.ibm.com/products/code-engine, accessed: July 2025.

[8] "Openwhisk: Open source serverless cloud platform," https://openwhisk.apache.org/, accessed: July 2025.

[9] "Knative documentation," https://knative.dev/docs/, accessed: July 2025.

[10] X. Chai, Y. Fu, J. Zhou, Y. Zhao, D. Li, and Y. Li, "Fork in the road: Reflections and optimizations for cold start latency in production serverless systems," in *Proc. OSDI*, 2025.

[11] D. Meisner, B. Gold, and T. Wenisch, "The PowerNap Server Architecture," *ACM Trans. Comput. Syst.*, vol. 29, no. 1, pp. 1–24, 2011.

[12] P. D. Welch, "On a generalized M/G/1 queuing process in which the first customer of each busy period receives exceptional service," *Oper. Res.*, vol. 12, no. 5, pp. 736–752, 1964.

[13] J. R. Artalejo, A. Economou, and M. J. Lopez-Herrero, "Analysis of a multiserver queue with setup times," *Queueing Syst.*, vol. 51, no. 1-2, pp. 53–76, 2005.

[14] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," *Perf. Eval.*, vol. 67, no. 11, pp. 1123–1138, 2010.

[15] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf, "Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward," *Queueing Syst.*, vol. 77, pp. 177–209, 2014.

[16] T. Phung-Duc, "Exact solutions for M/M/c/setup queues," *Telecommun. Syst.*, vol. 64, no. 2, pp. 309–324, 2017.

[17] J. K. Williams, M. Harchol-Balter, and W. Wang, "The M/M/k with deterministic setup times," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, Article No. 56, pp. 1–45, 2022.

[18] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perf. Eval.*, vol. 67, no. 11, pp. 1155–1171, 2010.

[19] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *ACM Trans. Comput. Sys.*, vol. 30, no. 4, pp. 1–26, 2012.

[20] N. Carlsson and D. Eager, "Optimized dynamic cache instantiation and accurate LRU approximations under time-varying request volume," *IEEE Trans. on Cloud Computing*, vol. 11, pp. 779–797, Jan/Mar. 2023.

[21] G. Dán and N. Carlsson, "Dynamic content allocation for cloud-assisted service of periodic workloads," in *Proc. IEEE INFOCOM*, 2014.

[22] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. IEEE INFOCOM*, 2013.

[23] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of Web server replicas," in *Proc. IEEE INFOCOM*, 2001.

[24] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. on Networking*, vol. 8, no. 5, pp. 568–582, Oct. 2000.

[25] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *Proc. IEEE INFOCOM*, 2001.

[26] P. Radoslavov, R. Govindan, and D. Estrin, "Topology-informed internet replica placement," *Computer Communications*, vol. 25, no. 4, pp. 384–392, Mar. 2002.

[27] C. W. Cameron, S. H. Low, and D. X. Wei, "High-density model for server allocation and placement," in *Proc. ACM SIGMETRICS*, 2002.

[28] F. Chen, R. K. Sitaraman, and M. Torres, "End-user mapping: Next generation request routing for content delivery," in *Proc. ACM SIGCOMM*, 2015, pp. 167–181.

[29] L. Wang, V. Pai, and L. Peterson, "The effectiveness of request redirection on cdn robustness," in *Proc. OSDI*, 2002.

[30] Y. Song, T. Wo, R. Yang, Q. Shen, and J. Xu, "Joint optimization of cache placement and request routing in unreliable networks," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 168–178, 2021.

[31] Y. Chawathe, S. McCanne, and E. Brewer, "Rmx: Reliable multicast in heterogeneous environments," in *Proc. IEEE INFOCOM*, 2000.

[32] G.-I. Kwon and J. W. Byers, "Roma: Reliable overlay multicast with loosely coupled tcp connections," in *Proc. IEEE INFOCOM*, 2004.

[33] S. Ganguly, A. Saxena, S. Bhatnagar, R. Izmailov, and S. Banerjee, "Fast replication in content distribution overlays," in *IEEE INFOCOM*, 2005.

[34] X. Xia, F. Chen, Q. He, J. C. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2020.

[35] J. Zerwas, I. Poese, S. Schmid, and A. Blenk, "On the benefits of joint optimization of reconfigurable cdn-isp infrastructure," *IEEE Trans. on Network and Service Management*, vol. 19, no. 1, pp. 158–173, 2021.

[36] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, 2019, pp. 10–18.

[37] ——, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[38] V. Farhadi, F. Mehmeti, T. He, T. F. La Porta *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. on Networking*, vol. 29, no. 2, pp. 779–792, 2021.

[39] R. Li, Z. Zhou, X. Zhang, and X. Chen, "Joint application placement and request routing optimization for dynamic edge computing service management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4581–4596, 2022.

[40] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *IEEE INFOCOM*, 1997.

[41] K. L. Johnson, J. F. Carr, M. S. Day, and F. Kaashoek, "The measured performance of content distribution networks," *Computer Communications*, vol. 24, no. 2, Feb. 2001.

[42] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, "Clustering and server selection using passive monitoring," in *Proc. IEEE INFOCOM*, 2002.

[43] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM*, 2002.

[44] J. M. Almeida, D. L. Eager, M. K. Vernon, and S. J. Wright, "Minimizing delivery cost in scalable streaming content distribution systems," *IEEE TMM*, vol. 6, no. 2, Apr. 2004.

[45] N. Carlsson and D. L. Eager, "Server selection in large-scale video-on-demand systems," *ACM Trans. on Multimedia Computing, Communications, and Applications*, vol. 6, no. 1, pp. 1:1–1:26, Feb. 2010.

[46] Z. Fei, M. H. Ammar, and E. W. Zegura, "Multicast server selection: Problems, complexity and solutions," *IEEE JSAC*, vol. 20, no. 7, 2002.

[47] M. Guo, M. H. Ammar, and E. W. Zegura, "Selecting among replicated batching video-on-demand servers," in *Proc. NOSSDAV*, 2002.

[48] N. Carlsson and D. Eager, "Content delivery using replicated digital fountains," in *Proc. IEEE/ACM MASCOTS*, Aug. 2010.

[49] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *Proc. IEEE INFOCOM*, 2003.

[50] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of TTL-based Cache Networks," in *Proc. Valuetools*, 2012.

[51] N. Carlsson, D. Eager, A. Gopinathan, and Z. Li, "Caching and optimized request routing in cloud-based content delivery systems," *Performance Evaluation*, vol. 79, pp. 38–55, 2014.

[52] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, Sept. 2002.

[53] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *Proc. IEEE INFOCOM*, 2010.

Niklas Carlsson is a Professor at Linköping University, Sweden. He received his M.Sc. degree in Engineering Physics from Umeå University, Sweden, and his Ph.D. in Computer Science from the University of Saskatchewan, Canada. He has previously worked as a Postdoctoral Fellow at the University of Saskatchewan, Canada, and as a Research Associate at the University of Calgary, Canada. His research interests are in the areas of design, modeling, characterization, performance, and security of distributed systems and networks.

Derek Eager received the BSc degree in computer science from the University of Regina, Canada, and the MSc and PhD degrees in computer science from the University of Toronto, Canada. He is a professor in the Department of Computer Science, University of Saskatchewan, Canada. His research interests include the areas of performance evaluation, content distribution, and distributed systems and networks.

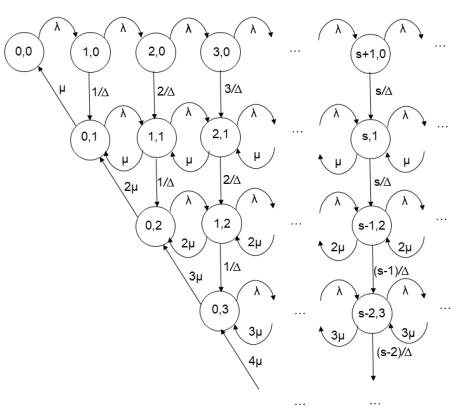


Fig. 14. State-Transition Diagram for Unlimited Server System.

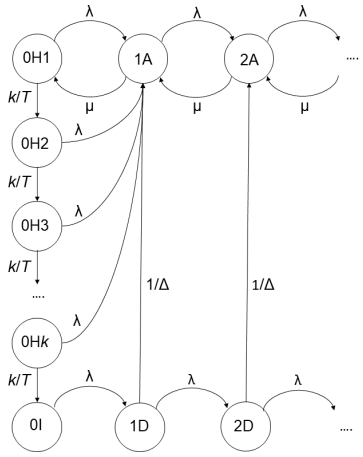


Fig. 15. State-Transition Diagram for M/M/1 + Setup Delay (exponential) + Delayed Off (Erlang) System.

APPENDIX

This appendix presents a summary of the unlimited server analysis results and M/M/1 server analysis results that we derive and present in [1]. (Note that the definition of "C" used in [1] differs slightly from that used here, and the expressions for "C" that we give here differ from those in [1] by a factor of S.) These results are used as a starting point for our analysis of *Reactive unlimited* server systems and *Dynamic M/M/1* server systems, respectively.

A. Unlimited Server Analysis

We consider a more general case where at most s server allocations, for some positive integer s, can be in progress at once. A state-transition diagram for this system is shown in Figure 14, where $\mu = 1/S$, S is the mean service time, and the request arrival rate is λ. Denote by $p_{i,k}$ the probability of the state $(i,\ k)$ with i waiting requests and k requests in service (equal to the number of active servers). Flow balance equations for this system are:

$$p_{i,0}\left(\lambda + \frac{\min[i,s]}{\Delta}\right) = p_{i-1,0}\lambda \qquad i > 0, \tag{39}$$

$$p_{0,k}(\lambda + k\mu) = p_{1,k-1}\left(\frac{1}{\Delta}\right) + p_{1,k}k\mu + p_{0,k+1}(k+1)\mu, \qquad k > 0, \tag{40}$$

and

$$p_{i,k}\left(\lambda + k\mu + \frac{\min[i,s]}{\Delta}\right) = p_{i+1,k-1}\left(\frac{\min[i+1,s]}{\Delta}\right) + p_{i-1,k}\lambda + p_{i+1,k}k\mu \qquad i > 0, k > 0. \tag{41}$$

It is straightforward to verify that the flow balance equations are satisfied by probabilities $p_{i,k} = p_{i,*}p_{*,k}$ where $p_{i,*}$ denotes the marginal probability of i waiting requests and $p_{*,k}$ denotes the marginal probability of k requests in service, with $p_{i,*}$ given by

$$p_{i,*} = \frac{\prod_{m=0}^{i}\frac{\lambda}{\lambda+\min[m,s]/\Delta}}{\sum_{n=0}^{s-1}\prod_{m=0}^{n}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{1}{1-\lambda/(\lambda+s/\Delta)}\right)}, \qquad i \geq 0 \tag{42}$$

and $p_{*,k}$ given by

$$p_{*,k} = \frac{(\lambda/\mu)^k e^{-\lambda/\mu}}{k!}, \qquad k \geq 0. \tag{43}$$

From Equation (42),

$$\sum_{i=0}^{\infty} ip_{i,*} = \frac{\sum_{i=0}^{s-1} i\prod_{m=0}^{i}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{s}{1-\frac{\lambda}{\lambda+s/\Delta}} + \frac{\lambda/(\lambda+s/\Delta)}{\left(1-\frac{\lambda}{\lambda+s/\Delta}\right)^2}\right)}{\sum_{n=0}^{s-1}\prod_{m=0}^{n}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{1}{1-\lambda/(\lambda+s/\Delta)}\right)}. \tag{44}$$

Applying Little's Law, and incorporating the service time once a request acquires a server, R is given by

$$R = \frac{\sum_{i=0}^{s-1} i\prod_{m=0}^{i}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{s}{1-\frac{\lambda}{\lambda+s/\Delta}} + \frac{\lambda/(\lambda+s/\Delta)}{\left(1-\frac{\lambda}{\lambda+s/\Delta}\right)^2}\right)}{\lambda\left(\sum_{n=0}^{s-1}\prod_{m=0}^{n}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{1}{1-\lambda/(\lambda+s/\Delta)}\right)\right)} + 1/\mu. \tag{45}$$

Next, C is given by

$$\begin{aligned} C &= \lambda S + \sum_{i=0}^{\infty}\min[i,s]p_{i,*} \\ &= \lambda S + \frac{\sum_{i=0}^{s-1} i\prod_{m=0}^{i}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{s}{1-\frac{\lambda}{\lambda+s/\Delta}}\right)}{\sum_{n=0}^{s-1}\prod_{m=0}^{n}\frac{\lambda}{\lambda+m/\Delta} + \left(\prod_{m=0}^{s}\frac{\lambda}{\lambda+m/\Delta}\right)\left(\frac{1}{1-\lambda/(\lambda+s/\Delta)}\right)}. \end{aligned} \tag{46}$$

For the special case of $s = 1$, these expressions reduce to $C = \lambda(S + \Delta/(1 + \lambda\Delta))$, and a mean response time R of $\Delta + S$. Note that with $s = 1$, this policy has the same mean response time as with a separate server per request, but lower server usage. This can be explained by the efficiency that results from taking a newly-free existing server for a waiting request instead of always requiring a new server.

B. Analysis for M/M/1 + Setup Delay (Exponential) + Delayed Off (Erlang)

A state-transition diagram for this system is shown in Figure 15. Here each state is labeled by the number of client requests present at the server, followed by "A" (server is active processing requests), "D" (server is in setup delay), "I" (server is deallocated), or "Hj" for integer j between 1 and k (server is the j'th stage of the Erlang-distributed "holding on" / "delayed off" period). The feasible states are states iA ($i \geq 1$), iD

($i \geq 1$), 0I, and 0Hj ($1 \leq j \leq k$), with state transition rates as shown in the figure.

Denote the steady-state probability of the state with label s by p_s. Assuming stability, we must have

$$\sum_{i=1}^{\infty} p_{i\mathrm{A}} = \frac{\lambda}{\mu} \Leftrightarrow \sum_{j=1}^{k} p_{0\mathrm{H}j} + p_{0\mathrm{I}} + \sum_{i=1}^{\infty} p_{i\mathrm{D}} = 1 - \frac{\lambda}{\mu}. \quad (47)$$

We can express the $p_{0\mathrm{H}j}$ and $p_{i\mathrm{D}}$ probabilities in terms of $p_{0\mathrm{I}}$, allowing the solution for $p_{0\mathrm{I}}$ using Equation (47). From flow balance, we have

$$p_{0\mathrm{H}k}(k/T) = p_{0\mathrm{I}}\lambda \quad (48)$$

and

$$p_{0\mathrm{H}j}(k/T) = p_{0\mathrm{H}(j+1)}(\lambda + k/T) \qquad 1 \leq j < k, \quad (49)$$

yielding

$$p_{0\mathrm{H}j} = p_{0\mathrm{I}} \left(\frac{\lambda T}{k}\right) \left(\frac{\lambda + k/T}{k/T}\right)^{k-j} \qquad 1 \leq j \leq k. \quad (50)$$

From Equation (50) we get:

$$\begin{aligned} \sum_{j=1}^{k} p_{0\mathrm{H}j} &= p_{0\mathrm{I}} \sum_{j=1}^{k} \left(\frac{\lambda T}{k}\right) \left(\frac{\lambda + k/T}{k/T}\right)^{k-j} \\ &= p_{0\mathrm{I}} \left(\frac{\lambda T}{k}\right) \left(\frac{1 - (\lambda T/k + 1)^k}{1 - (\lambda T/k + 1)}\right) \\ &= p_{0\mathrm{I}} \left((\lambda T/k + 1)^k - 1\right). \end{aligned} \quad (51)$$

We also have

$$p_{1\mathrm{D}}(\lambda + 1/\Delta) = p_{0\mathrm{I}}\lambda \quad (52)$$

and

$$p_{(i+1)\mathrm{D}}(\lambda + 1/\Delta) = p_{i\mathrm{D}}\lambda \qquad i \geq 1, \quad (53)$$

yielding

$$p_{i\mathrm{D}} = p_{0\mathrm{I}} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^{i} \qquad i \geq 1. \quad (54)$$

From Equation (54) we get:

$$\sum_{i=1}^{\infty} p_{i\mathrm{D}} = p_{0\mathrm{I}} \sum_{i=1}^{\infty} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^{i} = p_{0\mathrm{I}} \frac{\lambda/(\lambda + 1/\Delta)}{1 - \lambda/(\lambda + 1/\Delta)} = p_{0\mathrm{I}}\lambda\Delta. \quad (55)$$

Using Equations (51) and (55) to substitute into (47) gives:

$$p_{0\mathrm{I}} = \frac{1 - \lambda/\mu}{(\lambda T/k + 1)^k + \lambda\Delta}. \quad (56)$$

Denote by p_n ($n \geq 1$) the steady-state probability of n client requests being present at the server, i.e. the sum of $p_{n\mathrm{A}}$ and $p_{n\mathrm{D}}$. From flow balance,

$$\mu(p_1 - p_{1\mathrm{D}}) = \lambda \left(\sum_{j=1}^{k} p_{0\mathrm{H}j} + p_{0I}\right). \quad (57)$$

Substitution from (51), (54), and (56) gives

$$p_1 = \left(\frac{\lambda}{\mu}(\lambda T/k + 1)^k + \frac{\lambda}{\lambda + 1/\Delta}\right) \left(\frac{1 - \lambda/\mu}{(\lambda T/k + 1)^k + \lambda\Delta}\right). \quad (58)$$

Again, applying flow balance,

$$\mu(p_i - p_{i\mathrm{D}}) = \lambda p_{i-1} \qquad i \geq 2, \quad (59)$$

yielding, for all $i \geq 1$,

$$p_i = \left(\left(\frac{\lambda}{\mu}\right)^i (\frac{\lambda T}{k} + 1)^k + \sum_{l=1}^{i} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^l \left(\frac{\lambda}{\mu}\right)^{i-l}\right) \times \times \left(\frac{1 - \lambda/\mu}{(\frac{\lambda T}{k} + 1)^k + \lambda\Delta}\right). \quad (60)$$

Considering now the mean number of requests in the system $\sum_{i=1}^{\infty} i p_i$, where p_i is given by Equation (60), note that

$$\sum_{i=1}^{\infty} i(\lambda/\mu)^i = \frac{\lambda/\mu}{(1 - \lambda/\mu)^2} \quad (61)$$

and

$$\begin{aligned} \sum_{i=1}^{\infty} i \sum_{l=1}^{i} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^l \left(\frac{\lambda}{\mu}\right)^{i-l} &= \sum_{i=1}^{\infty} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^i \sum_{l=0}^{\infty} (l + i) \left(\frac{\lambda}{\mu}\right)^l \\ &= \sum_{i=1}^{\infty} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^i \left(\sum_{l=0}^{\infty} l \left(\frac{\lambda}{\mu}\right)^l + i \sum_{l=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^l\right) \\ &= \frac{\lambda/\mu}{(1 - \lambda/\mu)^2} \sum_{i=1}^{\infty} \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^i + \frac{1}{1 - \lambda/\mu} \sum_{i=1}^{\infty} i \left(\frac{\lambda}{\lambda + 1/\Delta}\right)^i \\ &= \frac{\lambda/\mu}{(1 - \lambda/\mu)^2}(\lambda\Delta) + \frac{\lambda\Delta(1 + \lambda\Delta)}{1 - \lambda/\mu}. \end{aligned} \quad (62)$$

Note that in the first line of (62), the original double summation is rewritten to group together all of the resulting terms that include the same power of $\lambda/(\lambda + 1/\Delta)$ as one of the factors. Applying (61) and (62) with (60), the mean number of requests in the system is given by

$$\begin{aligned} \sum_{i=1}^{\infty} i p_i &= \left(\frac{\lambda/\mu}{(1 - \lambda/\mu)^2} \left((\frac{\lambda T}{k} + 1)^k + \lambda\Delta\right) + \frac{\lambda\Delta(1 + \lambda\Delta)}{1 - \lambda/\mu}\right) \times \\ &\qquad \times \left(\frac{1 - \lambda/\mu}{(\frac{\lambda T}{k} + 1)^k + \lambda\Delta}\right) \\ &= \frac{\lambda/\mu}{1 - \lambda/\mu} + \frac{\lambda\Delta(1 + \lambda\Delta)}{(\frac{\lambda T}{k} + 1)^k + \lambda\Delta}. \end{aligned} \quad (63)$$

From Little's Law, the mean response time R is given by

$$R = \frac{S}{1 - \lambda S} + \frac{\Delta(1 + \lambda\Delta)}{(\frac{\lambda T}{k} + 1)^k + \lambda\Delta}. \quad (64)$$

C is given by

$$C = 1 - \frac{1 - \lambda S}{(\lambda T/k + 1)^k + \lambda\Delta}. \quad (65)$$

Special cases:

1) T is exponentially distributed ($k = 1$):

$$R_{k=1} = \frac{S}{1 - \lambda S} + \frac{\Delta(1 + \lambda\Delta)}{\lambda T + 1 + \lambda\Delta}; \quad C_{k=1} = 1 - \frac{1 - \lambda S}{\lambda T + 1 + \lambda\Delta}. \quad (66)$$

2) T is deterministic ($k \to \infty$):

$$R_{k\to\infty} = \frac{S}{1 - \lambda S} + \frac{\Delta(1 + \lambda\Delta)}{e^{\lambda T} + \lambda\Delta}; \quad C_{k\to\infty} = 1 - \frac{1 - \lambda S}{e^{\lambda T} + \lambda\Delta}. \quad (67)$$

Key insight (M/M/1 case): Significant benefit of using deterministic delayed-off periods compared to when the delayed-off periods are exponentially distributed.