Impact of Load Sharing on Provisioning Services with Consistency Requirements

Daniel Villela* Vishal Misra† Dan Rubenstein* Sambit Sahu‡

* Dept. of Electrical Engineering
Columbia University
New York, NY 10027
{dvillela,danr}@ee.columbia.edu

† Dept. of Computer Science
Columbia University
New York, NY 10027
misra@cs.columbia.edu

‡ Network Software and Servers
IBM T. J. Watson Research Center
Hawthorne, NY 10532
sambits@us.ibm.com

Abstract—
Providers of services such as online auctions must provision servers to respond quickly to users' requests. Assigning each service to its own set of servers can result in some servers being overloaded while others remain underloaded. In contrast, load sharing spreads all services across the sets of servers, allowing each request to each be serviced from a choice of multiple servers, relieving the busiest servers. Sharing arbitrarily across servers, however, can be costly because of the need to maintain consistency of the service across these servers. In this work we model servicing systems with consistency requirements and analyze the impact of load sharing. Our analysis of greedy algorithms provides upper bounds of the response times, and shows that greedy algorithms can reduce the busiest server's intensity to be very close to the average service intensity - a near-optimal result. We also reveal a surprising result that, when servers' average loads are equal but fluctuate over time, wthe average response time can be reduced by sharing load.

I. Introduction

The success of various online services such as auctioning systems (e.g, eBay) and e-Commerce systems (e.g, Amazon.com) hinges on their ability to respond quickly to user requests, regardless of demand. Hence, the providers of these kinds of services must provision their servers so that response times remain low even wheen demand is high. The total demand at a serving system is comprised of an aggregate of demands from multiple *servicing instances*. For example, an auction site offers several auctions simultaneously. Each auction is an individual serving instance and has its own individual demand that fluctuates over time. If the auction is served by a single server, when the demand spikes, the response time will drop.

This paper investigates the use of *load sharing* across servers. When servers load share, a single servicing instance can be served by multiple servers, and each server can simultaneously host (part of) several different instances. Load sharing permits a more equitable distribution of load across servers. However, there is a consistency cost: the set of servers hosting a common instance must keep certain information that they share consistent at all times. For example, an auction site may partition the participants of a popular auction across servers, such that each server is responsible for receiving bids from the participants and informing them of the current highest bid. To do this, the servers must keep a consistent view of the current highest bid. If an instance is replicated over too many servers, then the cost of maintaining consistency can outweigh the savings gained from distributing the load of requests. *Our goal is to find a solution that minimizes the load placed on the heaviest loaded server.*

In actual servicing systems, server infrastructures are comprised of tens to tens of thousands of servers [**?**, **?**, **?**]. Load sharing is indeed applied to these server infrastructures. However, to the best of our knowledge, the load sharing strategy is *ad hoc*. Hence, investigating how servicing instances should be assigned to servers to reduce response times still requires attention. The fundamental questions that we address in this work are:

- How can load sharing reduce the response times for services which, when distributed across multiple servers, must be kept consistent?
- Can load sharing reduce response times when instance's demands are already distributed such that loads across servers are approximately equal? And, if it can, what should be taken into account to determine whether load sharing should be applied in that case?

We study simple, greedy algorithms that determine how load is to be "shared" in a serving system where consistency must be maintained and there is a cost for this maintenance. We model this class of servicing systems and analyze the intensity at the busiest server. We propose a greedy algorithm that takes as input the set of servers and the set of instances and outputs an assignment of the instances to the servers. We show analytically that the greedy algorithm yields an allocation with a maximum serving intensity that is within a constant factor of the optimal maximum intensity. We then investigate a number of cases empirically that suggest these algorithms produce configurations in which the maximum intensity is close to the initial average demand, and is usually significantly smaller than the maximum intensity in a configuration without load sharing, when each instance is served by a single server. This is especially important because the problem is shown to be NP–hard.

Next, we extend our model for the case in which instances' time-average demands are equal, but fluctuate independently over small time scales. One might posit that load sharing offers no benefit since average demands are equal to begin with, and imposing load sharing would add a consistency cost. Our results show, however, that in fact, average response times can still be reduced by load sharing because the variance of loads at

This material was supported in part by the National Science Foundation under CAREER Award No. ANI-0133829 and NSF ITR-0325495 Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

servers is reduced.

The remainder of the paper is structured as follows. In Section II, we overview related work. In Section III, we describe our model for servicing systems in which demands fluctuate at large time scales. In Section IV, we formally define two greedy algorithms and analyze their performance. In Section V, we describe an extension of the model that takes into account fluctuations of demands that occur in small time scales. We conclude in Section VI.

II. Related Work

The objective in the classical scheduling problem [**?**] is to minimize the completion time for running a set of tasks whose running times are known by distributing them over a set of machines. This problem is related to our problem, by mapping loads of servicing instances to running times and machines to servers. [1] However, in many systems, however, tasks cannot be fragmented, or if they can, there is a cost for fragmenting them. Previous algorithmic approaches for task scheduling across machines [?, **?**, **?**, **?**], for provisioning server loads in [**?**], and for traffic engineering using interior gateway routing protocols [**?**] do not consider fragmentation costs in their models. The optimization approaches in [**?**, **?**, **?**] and the performance study in [**?**] also differ from our work in this same fundamental aspect.

The goals and assumptions in previous work that study systems that require consistency when provisioning serving resources differ from ours. For instance, the work in [**?**] describes the problem of minimizing the number of copies in a caching system. In [**?**], Coffman *et al.* study the number of replications of database entries to maximize the fraction of server processing needed to service only queries as a function of the arrival rate of database updates. Our goal to minimize the maximum intensity at servers is then clearly different than the goals in [?] and [?]. The works in [**?**, **?**, **?**, ?] present queueing–theoretic models to describe the dependencies in a set of database queries by the constraints that a query must wait for write tasks to be finished. In our work, consistency is taken into account without focusing on timing issues such as this waiting constraint for database queries. Because of these assumptions, these models do not suit our purposes here.

Finally, current provisioning experiences for representative large–scale services are reported in [?, ?, ?] that describe server architectures of commercial services employing from hundreds to thousands of servers for applications such as search engines (Google [?] and Hotbot [?]) or online massive games [?]. Load sharing, among other features, is then cited for scalability, but none of these works evaluates its impact.

III. Model and Objectives

Our goal is to investigate an alternative to the provisioning policy in which any request can only be serviced by a single server. In this section, we assume that the service demand for each servicing instance is constant, or at least fluctuates at a rate lower than the measurement rate used by the service provider to give estimates of those demands. This assumption is not unreasonable for many services whose demands are correlated to routine daily activities that are well known [**?**] (e.g., consistent peaks at evenings) and typically a peak period may last for a long time. Another example is that of a ticketing system demand for booking tickets can be quite high for an hour and slows down afterwards. In Section V, we present an extension of the model that deals with the case that demand fluctuates significantly within a measurement interval.

A. Modeling a servicing system

We model a provider as a system consisting of a set of m servers, $\mathcal{S} = \{s_1, s_2, ..., s_m\}$, that manage the servicing of m servicing instances in the set $\mathcal{A} = \{a_1, a_2, ..., a_m\}$. Examples for such service instances include item auctions (auctioning system), multiple online games, or item sales (e-commerce).

Requests for instances are classified into two classes of requests. A first class of requests, *read* requests, consists of requests that access content on the server without altering it. In the online auction example, a read request can be the description of an auctioned item or the current highest existing bid. A second class of requests, *write* requests, consists of requests that alter the content being served. Examples are a seller submitting new information on an auctioning item, or a buyer submitting a bid. Each instance's demand can be described by an intensity $\gamma(a_i) = \gamma_i$ due to read requests plus an intensity $v(a_i) = v_i = f_i\gamma(a_i)$, $1 \leq i \leq l$, due to write requests.[2] A *consistency requirement* is imposed for every instance a_i such that a modification to a_i at one server must be immediately applied at every server servicing that instance.

When load sharing is implemented, each service instance a_i must be *fragmented* into $z(a_i) = z_i$ fragments. Then, these fragments must be assigned to a subset of the servers in the system. Note that when load sharing is not implemented, we say that a *static provisioning* policy is used, and $\forall a_i \in \mathcal{A}$, $z_i = 1$.

When fragments sizes can be arbitrary, we say the fragmentation is *fluid*, and an assignment can be viewed as the "pouring" of an instance into each server. We also consider the case where an instance must be split into equal size fragments. Hence, the intensity due to instance a_i's read requests at each of the servers is γ_i/z_i, since z_i is the number of servers servicing instance a_i and intensities are divided into equal portions.

A server's intensity describes how much of the server's processing is required in a unit of time. This is naturally a function of the read and write intensities of instances placed on a server. The intensity $\rho(s_j)$ at server s_j is given by the sum of fragments of read intensity assigned to s_j plus the sum of write intensities on server s_j.

To enable a fair comparison with static provisioning, we assume that there are m instances being serviced by m servers throughout the rest of this paper. This is a necessary condition for static provisioning, but not for load sharing.

B. Objective: minimizing maximum intensity

We now define the response time of a server s as a function $T(s)$. In particular we focus on non–linear, non–decreasing functions $T(s)$ that are function of $\rho(s)$. A processor sharing model, for instance, has been used to describe servicing of

[1] Our problem is also related to bin packing, since scheduling is related to bin packing [**?**].

[2] We generally assume that $0 < f_i \leq 1$, but this condition is not necessary.

tasks in servers [**?**]. For a processor sharing system, the function $T(s) = \frac{1}{c-\rho(s)}$, where c is a constant describing a server's capacity, is the response time function for a unit size job. The capacity of a server measures the maximum amount of work processed by the server in one unit of time. The performance metric to be assessed is the maximum response time across all servers.

Since the response time is a non–decreasing function of the intensity placed on servers, our objective is to minimize the maximum intensity across servers. The largest server intensity in the solution is then given by ρ_{OPT}. Formally,

$$\rho_{OPT} = \min \max_{s \in \mathcal{S}} \rho(s) \quad (1)$$

Furthermore, in the case of systems with heterogeneous capacities a provider must maximize the minimum remaining capacity given by the difference between the capacity of a server and its intensity.[3]

The problem of distributing fragments onto servers in order to minimize the maximum intensity can be restricted to an instance of the classic scheduling problem. This restriction reveals the problem to be in the class of NP–complete problems (details in Section IV). Therefore we study algorithms based on simple heuristics.

Let an algorithm A result in a maximum intensity ρ_A across servers. Our goal is to compare the result for an algorithm A compared to the optimal result ρ_{OPT} given by the minimum maximum intensity. We generally establish bounds that determine how large the largest intensity obtained by A is compared to ρ_{OPT}.

We assume, without loss of generality, that a static provisioning configuration is an input to the problem. It is useful to define the original instance of a server s_j to be the instance present in that server in the static provisioning configuration. Even when already having a configuration based on load sharing a provider can still compute a new configuration based on load sharing using the original item configuration corresponding to the configuration of static provisioning as an input.

We remark that the question about how to maintain consistency, rather than the impact of a policy such as load sharing given the consistency factor, is legitimate but is not the focus of our study. Answering such question is indeed orthogonal to our work.

IV. Algorithms for load–sharing

In this section we state two algorithms based on a greedy policy to reduce the intensity of the busiest servers in provisioning a service with consistency requirements. We can reduce the problem studied here to a well-known, NP–complete problem, classical scheduling, to show that the problem is NP–hard. This fact makes the study of greedy algorithms especially relevant. Let us take a particular instance of our problem in which the configuration that minimizes the maximum intensity is such that the sum of fragment sizes at every server is equal to $\rho^{'}$ and the intensity ρ_{OPT} equal at all servers $s \in \mathcal{S}$. If an instance admits this configuration, then it is optimal. This is true because any fragment made smaller at some server s means that a larger fragment exists in another server other than s, hence the intensity in that second server is higher than ρ_{OPT}. Therefore, this configuration indeed minimizes the maximum intensity found to be ρ_{OPT}. This instance is also a restriction from our problem since we know the fragmentation factor z_i for all instances $a_i \in \mathcal{A}$, instead of having to determine z_i, $1 \leq i \leq m$ as part of the problem. In that case, let us place for all instances fragments of arbitrary sizes with the constraint that the sum of z_i fragment sizes of any instance a_i is γ_i (as given in the problem), but such that the sum of fragment sizes placed at every server is $\rho^{'}$. The problem becomes minimizing the quantities $\tau_j = \rho(s_j) - \rho^{'}$, which is the sum of write intensities at server $s_j \in \mathcal{S}$. Therefore placing fragments is equivalent to adding items (fragments of instances) whose sizes are given by the write intensities v_i, since read intensities sum up to $\rho^{'}$. Thus, if we solve the problem of finding the configuration that minimizes the largest sum of write intensities in every server we have also resolved our problem. This corresponds to the scheduling problem of $\sum_{i=1}^{m} z_i$ items of size v_j, $1 \leq j \leq \sum_{i=1}^{m} z_i$, over a set of m servers $\mathcal{S}$ which is a well-known, NP–hard problem. Therefore, the problem here is unlikely to admit a solution from a polynomial time algorithm, under the conjecture that $P \neq NP$. It is interesting to note that when $f_i = 0$, for any $a_i \in \mathcal{S}$, the problem becomes simple to solve. In this case, there are no write intensities and the read intensity can be placed in arbitrary sizes such that the average intensity at every server reaches the average level.

We proceed next with useful definitions for general algorithms and simple bounds.

A. Useful definitions and facts

Definition A *movement* from s to $s^{'}$ is the transfer of a fragment of an instance's read intensity from a server s to server $s^{'}$.

A movement from s to $s^{'}$ causes an additional write intensity at server $s^{'}$ equal to the write intensity of the instance's passed from server s, as previously explained.

Definition An end configuration is said to be a *definitely final* configuration if no possible movement can exist from a server to another without increasing the largest intensity across servers.

In an optimal end configuration, there can be no movement of fragment from one server to another at the expense of increasing the largest intensity across all servers. Therefore, a desirable property in an algorithm is that it reaches a definitely final configuration.

Fact IV.1: For a definitely final configuration the difference between intensities of any pair of servers s_l and s_j, $\rho(s_l) \geq \rho(s_j)$, we have

$$\rho(s_l) - \rho(s_j) \leq v_{max},$$

where v_{max} is the largest fragmenting cost (largest writing intensity).

This gives important insight on why the variance of servers' intensities is reduced when applying the load sharing concept. Let us also state formally a useful observation that we often use in this work.

Fact IV.2: Let $\rho^* = \max\{\gamma_1 + v_1, \gamma_2 + v_2, ..., \gamma_m + v_m\}$ be the largest server intensity before changing the system balance

[3]Minimizing the maximum intensity also reduces the variance of intensities across servers, although minimizing variance is not the primary objective here.

into a load sharing configuration. We have simple bounds for ρ_{OPT} as follows.

$$\rho^* \geq \rho_{OPT} \geq \frac{1}{m}\sum_{i=1}^{m}(\gamma_i + v_i), \quad (2)$$

which simply states that the optimum lies between the average of intensities across all servicing instances and the largest server intensity.

Definition A *greedy policy* determines the movement of an instance's read intensity or a fragment of an instance's intensity into the server of least minimum intensity. Formally, the greedy policy specified that an instance a must be placed at s^*, such that $\rho(s^*) = \min_{s\in\mathcal{S}} \rho(s)$. This greedy policy can be generalized for a set of d fragments of a same instance to be placed in parallel into d servers with lowest intensities.

Let us also define $\bar{f} = \frac{\sum_{i=1} f_i\gamma_i}{\sum_{j=1}\gamma_j}$, the average of f_i weighted on γ_i, $1 \leq i \leq m$. This is useful to find the sum of intensities as a linear function of the sum of read intensities, since $\sum_{i=1}^{m}(\gamma_i + v_i) = \sum_{i=1}^{m}\gamma_i + \sum_{i=1}^{m} f_i\gamma_i = (1+\bar{f})\sum_{i=1}^{m}\gamma_i$. As a result of the fact above we also find for $\sum_{i=1}^{m}\gamma_i$:

$$\frac{1}{m}\sum_{i=1}^{m}(\gamma_i + v_i) = \frac{1}{m}\left(\sum_{i=1}^{m}\gamma_i + \bar{f}\sum_{i=1}^{m}\gamma_i\right) \leq \rho_{OPT}$$

$$\sum_{i=1}^{m}\gamma_i \leq \frac{m\rho_{OPT}}{1+\bar{f}}. \quad (3)$$

Let us also have for every server s_j its intensity $\rho(s_j)$ given by the sum w_j of fragments and the sum $h(s_j) = h_j$ of write intensities, $s_j \in \mathcal{S}$.

Next we use the greedy policy to construct algorithms.

B. Description of the greedy algorithms

Here we build simple heuristic–based algorithms that perform movements using the greedy policy. The key idea is to limit the number of fragments to not have an excessive total write intensity at servers.

Let us consider initially an algorithm which uses the even-size fragmentation model. Such model is useful because this kind of fragmentation can be easily implemented as a distribution action taken by a request dispatcher. The read requests of an instance a_i are directed to a single server at rate $1/z_i$ of the actual rate without any further complexity. As part of the algorithm a value k is the maximum value that any z_i can assume. In the input of the algorithm we have $\rho(s_j) = \gamma(a_j)(1+f_j)$, $1 \leq j \leq m$. The problem with passing a fragment from a server to another in only one direction, i.e., from a server of larger intensity to a server of smaller intensity is that the obtained configuration can have the maximum intensity larger than the previous one. In this case such movement should obviously be avoided. One way to improve the result is to permit an exchange of fragments such that not only a fragment is passed from the server of largest intensity to the ones of least intensity, but to also have fragments taken from the server of largest intensity and from the ones of least intensity (in a number of up to $k-1$) and passed one another. In this case the sum of fragment sizes generally is equal to the average of $\ell \leq k$ instances' read intensities. Pick the server with greatest intensity, say server s_i. Now taking a number ℓ of up to $k-1$ servers of smallest intensities, we proceed to the configuration that gives the smallest maximum intensity. One fragment should stay in the current server and the other $\ell - 1$ fragments are potentially placed in the servers with lowest intensities, i.e. according to the greedy policy. After re–computing the intensities at all other servers we pick the server with the current highest intensity. The procedure continues. In the particular case of $k = 2$, the algorithm consists of attempting a series of pairwise operations between the current server of largest intensity with the server of current smallest intensity. This is algorithm `GREEDY(k)`.

Another algorithm is `GREEDY-FLUID` which takes the fluid fragmentation model. The policy here is to equalize the intensity at a set of servers with movements always in the direction that follows from passing fragments from the server of largest intensity to the ones of least intensity. This takes advantage that fragments can be of arbitrary sizes. The idea comes from an analogy of pouring fluid from a vial to another until both of them reach equal levels of fluid. This policy as opposed to having exchanges of fragments from a server to another and the fluid fragmentation model as opposed to the even–size fragmentation model make this algorithm differ fundamentally from the previous one. It is necessary to evaluate, as part of the algorithm, the amount of fluid that should be transferred from a server s_i to the largest number of servers $s \in \mathcal{S}' \subset \mathcal{S}$, selected using the greedy policy, such that s_i's intensity decreases the most or another server, rather than s_i server, becomes the one with highest intensity. By taking advantage of the fluid model, as explained, intensities of all servers in $\mathcal{S}'$ are equal after a movement from s_i of instance a_i's intensity, i.e. $\rho(s_i) = \rho(s_j)$, $s_j \in \mathcal{S}'$. We remark that the write intensity $f_i\gamma_i$ is added to $\rho(s_j)$, if no portion of a_i's intensity is contained in server s_j. Next the algorithm proceeds to the server with highest intensity and repeats the procedure.

Fig. 1. Configurations obtained after movements given by the even–size fragmentation model (center) and the fluid fragmentation model (right). By contrast, a configuration without load sharing is shown to the left.

Figure 1 illustrates an example in which movements are performed across a set of servers $\{s_1, s_2, s_3\}$ for a set of instances $\{a_1, a_2, a_3\}$ under both fragmentation models. The bars' heights indicate the total intensity at each server. Shadowed rectangles indicate the portion of the intensity due to write requests, whereas white rectangles indicate the portion of intensity due to read requests. The configuration on left-hand side shows the original distribution in which no load sharing is performed. The configuration in the center shows the result after the two–way movements among servers s_1 and s_3 of fragments of sizes $\gamma(a_1)/2$ and $\gamma(a_3)/2$ each. The configuration in the right–hand side shows the result after a fragment of a_1 of size such that the intensities at both servers s_1 and s_3 are equal. Note that on both movements the cost of write requests is added to server s_3. Nevertheless, the maximum intensity is reduced from

the original configuration in both cases.

In particular, for the server of largest intensity ρ_G for `GREEDY(k)` (ρ_{GF} for `GREEDY-FLUID`)we can use for convenience the notation w_G (w_{GF}) for the sum of fragment sizes and h_G (h_{GF}) for the sum of write intensities.

Finally, the complexity of `GREEDY(k)` and `GREEDY-FLUID` is of the order of O(m log(m)) since the algorithms require sorting instance's intensities.

Next, we study the worst case analysis of the maximum intensity provided by these algorithms, i.e., how bad they can perform compared to an algorithm that minimizes the maximum intensity.

C. Analysis of `GREEDY(k)`

Lemma IV.3: In `GREEDY(k)` the total write intensity $\sum_{s\in\mathcal{S}} h(s)$ placed on servers is bounded as follows

$$\sum_{s\in\mathcal{S}} h(s) \leq \frac{mk\rho_{OPT}\bar{f}}{1+\bar{f}}. \tag{4}$$

Proof: Here the fragmentation of load from any instance, say a_i, generates at most $k-1$ fragments to be placed in servers other than server s_i. In order to have an upper bound on the total cost due to write intensities in the output of the algorithm, the $k-1$ write intensities should be counted for all m instances plus the write intensities that the system contains in static provisioning. Hence, $\sum_{s\in\mathcal{S}} h(s) \leq k\sum_{i=1}^{m} v_i = k\sum_{i=1}^{m} f_i\gamma_i$. We use then (3), to find (4):

$$\sum_{s\in\mathcal{S}} h(s) \quad \leq \quad k\sum_{i=1}^{m} f_i\gamma_i = k\bar{f}\sum_{i=1}^{m}\gamma_i \leq \frac{mk\rho_{OPT}\bar{f}}{1+\bar{f}}.$$

■

Theorem IV.4: Let ρ_G be the maximum intensity obtained under the `GREEDY(k)` algorithm. Then we have the following two bounds. First,

$$\rho_G \leq \rho_{OPT}\left(1+2k\frac{\bar{f}}{1+\bar{f}}\right) = \rho_{GB2}. \tag{5}$$

and, second, another bound

$$\rho_G \leq \rho_{OPT}\left(2+\frac{(k-1)\bar{f}}{1+\bar{f}}-\frac{1}{m}\right) = \rho_{GB1}. \tag{6}$$

Hence, $\rho_G \leq \min(\rho_{GB1}, \rho_{GB2})$.

Proof: We start proving (5). We use the fact that in the end configuration we cannot have any further movement for any l up to k. Therefore, for any number $l-1$ of servers, $1 \leq l \leq k$, an exchange with the one with largest intensity would take the read intensities at each of the l servers to a fraction of $1/l$ of the read intensities, i.e. an average, but write intensities would compound in manner that can exceed the largest intensity ρ_G. If such condition holds for any group of $l-1$ servers to be grouped with the server of intensity ρ_G, then no movement can be performed. The condition for not having any movement is formally stated

$$\begin{aligned} w_G + h_G \quad \leq \quad & \sum_{\ell=1}^{l-1}\left(\frac{w_\ell}{l}+h_\ell\right) + w_G/l + h_G, \\ & \forall\{s_\ell\}_{\rho(s_\ell)<\rho_G} \subset \mathcal{S}, |\{s_\ell\}| = l-1, 2 \leq l \leq k. \end{aligned}$$

Now let us take $\lfloor (m-1)/(l-1)\rfloor$ groups of $l-1$ servers each plus the server of largest intensity. We rewrite the previous inequality and subsequently make an algebraic manipulation to derive the following:

$$\begin{aligned} w_G + h_G \quad \leq \quad & \sum_{\ell=1+g(l-1)}^{(g+1)(l-1)}\left(\frac{w_\ell}{l}+h_\ell\right) + w_G/l + h_G, \\ & \forall s_j \in \mathcal{S}, 2 \leq l \leq k, \\ & 1 \leq g \leq \lfloor (m-1)/(l-1)\rfloor. \\ \frac{(l-1)w_G}{l} \quad \leq \quad & \sum_{\ell=1+g(l-1)}^{(g+1)(l-1)}\left(\frac{w_\ell}{l}+h_\ell\right), \\ & \forall s_j \in \mathcal{S}, 2 \leq l \leq k, \\ & 1 \leq g \leq \lfloor (m-1)/(l-1)\rfloor. \end{aligned}$$

The quantities of w_ℓ and h_ℓ are unknown, but if we sum the read intensities placed on all servers it must equal the total read intensities over all instances. We know that the total write intensity at servers is not necessarily equal to the sum of write intensities over all instances, but certainly larger or equal to the sum of intensities over all instances and bounded as stated in Lemma IV.3. Therefore, summing up over all groups we also sum up the unknown variables w_ℓ into the sum of read intensities and sum of write intensities. Formally,

$$\frac{mw_G}{l} \leq \frac{1}{l}\sum_{i=1}^{m}\gamma_i + \sum_{g=1}^{\lfloor (m-1)/(l-1)\rfloor}\sum_{\ell=1+(g-1)l}^{g(l-1)} h_\ell$$

And more simply, $mw_G \leq \sum_{i=1}^{m}\gamma_i + l\sum_{s\in\mathcal{S}} h(s)$. This inequality is valid for $2 \leq l \leq k$. Therefore, among all values that l can assume, $l=2$ provides the tightest bound for w_G. Next, we simply use the bounds for both $\sum_{s\in\mathcal{S}} h(s)$ and ρ_{OPT} and algebraic manipulations for a bound for w_G:

$$\begin{aligned} mw_G \quad & \leq \quad \sum_{i=1}^{m}\gamma_i + 2k\sum_{i=1}^{m} f_i\gamma_i \\ mw_G \quad & \leq \quad \sum_{i=1}^{m}(\gamma_i + f_i\gamma_i) + (2k-1)\sum_{i=1}^{m} f_i\gamma_i \\ mw_G \quad & \leq \quad m\rho_{OPT} + (2k-1)\frac{\bar{f}m}{1+\bar{f}}\rho_{OPT} \\ w_G \quad & \leq \quad \rho_{OPT}\left(1+(2k-1)\frac{\bar{f}}{1+\bar{f}}\right) \end{aligned}$$

The inequality above permits us to finish proving (5). For $\rho_G = w_G + h_G$, we find

$$\rho_G \leq w_G + \sum_{i=1}^{m} f_i\gamma_i \leq \rho_{OPT}\left(1+2k\frac{\bar{f}}{1+\bar{f}}\right).$$

We proceed with proving the bound in (6). We utilize an argument very similar to the one formulated in [?]. Let the algorithm end with maximum intensity ρ_G. Considering that the last fragment adds a portion of intensity $y = x+v$ (x, read intensity and v, write intensity) added to a server we have that all servers' intensities are at least $\rho_G - y$. Otherwise the last fragment should

go to the server with intensity less than $\rho_G - y$. Hence, if we sum up intensities over all servers and discount the total write intensities, the result should be below the sum of read intensities over all instances. Hence we have

$$\begin{aligned}(\rho_G - y)m + y - \sum_{s\in\mathcal{S}} h(s) &\leq \sum_{i=1}^{m}\gamma_i \\ (\rho_G - y)m + y - \sum_{s\in\mathcal{S}} h(s) + \sum_{i=1}^{m} f_i\gamma_i &\leq m\rho_{OPT}\end{aligned} \tag{7}$$

Using the bound for $\sum_{i=1}^{m}\gamma_i$ from (3) and the result from (4), we find

$$(\rho_G - y)m + y - \frac{\bar{f}m(k-1)\rho_{OPT}}{1+\bar{f}} \leq m\rho_{OPT} \tag{8}$$

Finally, using the fact that $y \leq \rho_{OPT}$ into (8), we prove the bound in (6) and finish the proof of the theorem.

$$\rho_G \leq \rho_{OPT}\left(2 + \frac{(k-1)\bar{f}}{1+\bar{f}} - \frac{1}{m}\right) \tag{9}$$

■

In the case of $k = 1$, which means no fragmentation whatsoever, we have the familiar result for a greedy algorithm to the classical scheduling problem, stating $\rho_G \leq \rho_{OPT}(2 - 1/m)$.

D. Analysis of GREEDY-FLUID

Let us first define the maximum fragmentation cost $v_{max} = \max_{a_i\in\mathcal{A}} v_i$. In particular, if $\forall a_i \in \mathcal{A}$, $f_i = f$, then v_{max} corresponds to the cost of fragmenting the instance of largest intensity defined by γ_{max}, $v_{max} = f\gamma_{max}$.

Lemma IV.5: For GREEDY-FLUID the total sum of write intensities is bounded as follows

$$\sum_{s\in\mathcal{S}} h(s) \leq \sum_{i=1}^{m} f_i\gamma_i + (m-1)v_{max}. \tag{10}$$

Proof: Here fluid is poured onto $k^{(1)} \leq k$ servers, where $k^{(\ell)}$ denotes the number of pourings in the ℓ iteration. Similarly, let $\rho_G^{(\ell)}$ mean the largest intensity in the ℓ iteration. The first iteration lets $k^{(1)}$ to have equal intensities, thus up to $m-1-k^{(1)}$ servers' intensities can be above ρ_1. Hence, fluid is poured onto $k^{(2)}$ servers, $k^{(2)} \leq m-1-k^{(1)}$. When no more movements can be executed, we have $\sum_{i=1}^{\Delta} k^{(\ell)} = m-1$, where Δ is the total number of rounds of movements. Therefore we have movement of fragments causing additional write intensities of at most $m-1$ times and of the order smaller than or equal to $(m-1)v_{max}$. Finally, taking into account the initial sum of write intensities over all instances, $\sum_{i=1}^{m} f_i\gamma_i$, we have proven the lemma. ■

Theorem IV.6: For GREEDY-FLUID the maximum intensity ρ_{GF} is bounded as follows

$$\rho_{GF} \leq \rho_{GFB} = \rho_{OPT} + (2 - \frac{1}{m})v_{max}.$$

Proof: The proof starts with an argument similar to the one used to prove Theorem IV.4. We know that no movement can be performed in a very strict sense, i.e. if for a fragment of negligible size ϵ to be passed from the server of largest intensity to a server of smaller intensity the write intensity to be added to the one with smaller intensity causes the intensity to be larger than previously. Hence, we can use the formal statement of Fact IV.1, i.e. $\rho_{GF} - \rho_j \leq v_{max}$, $\forall s_j \in \mathcal{S}$. We then sum up over all servers $s_j \in \mathcal{S}$, such that we can relate the total read intensities and write intensities by the sum of read intensities over all instances and the bound found in Lemma IV.5, resulting,

$$m\rho_{GF} - \sum_{j=1}^{m}\rho_j = m\rho_{GF} - (\sum_{i=1}^{m}\gamma_i + \sum_{j=1}^{m} h_j) \leq mv_{max} \tag{11}$$

Using the bound in (10) and manipulating the expression, we derive

$$m\rho_{GF} - \sum_{i=1}^{j}(\gamma_i + f_i\gamma_i) - (m-1)v_{max} \leq m\rho_{GF} - \sum_{j=1}^{m}\rho_j$$

Finally, this permits us to find $\rho_{GF} \leq \rho_{OPT} + v_{max}\left(2 - \frac{1}{m}\right)$, which concludes the proof.

■

The importance of this bound is that ρ_{OPT} is without any multiplicative factor and the bound does not increase as m increases.

Corollary IV.7: When $f = 0$, $\rho_{GF} = \rho_{OPT}$, which indeed is an expected result.

E. Quantitative analysis

In this section we present various inputs and outputs for a service described by $m = 16$ instances and servers. We use as inputs the instances' intensities to be samples of either of three well-defined curves, a gaussian curve, a fast–decaying curve and a line. First, the intensity described by a gaussian function is $\Gamma\frac{e^{-(x-\bar{\gamma})^2/(2\sigma_\gamma^2)}}{\sigma_\gamma\sqrt{2\pi}}$, where x is equal to the server label i for any $s_i \in \mathcal{S}$, Γ $\bar{\gamma}$ and σ_γ are constants. The fast–decaying curve and the line are described by $\Gamma\frac{1}{x^{\alpha-1}}$ and $\Gamma\frac{x}{m}$, where α and Γ are constants and x is equal to the server label i for a server $s_i \in \mathcal{S}$. The server labels are given by an integer sequence from 1 to m.

The question on how to depict the bounds is not simple to answer since it is difficult to find the optimal configuration, hence finding the largest intensity in that configuration. A monte–carlo simulation needs a large number of runs in order to produce results close to optimal with high likelihood. In the cases considered here the number of runs necessary for accurate results is prohibitively large. The used approach is to observe $\zeta^{-1}(\rho_L)$, where ρ_L is the largest intensity in an output and ζ the function of ρ_{OPT} that expresses the bound (ζ^{-1} is the inverse function). We always have that $\rho_{OPT} \geq \zeta^{-1}(\rho_L)$, hence $\zeta^{-1}(\rho_L)$ is a lower bound of ρ_{OPT}. If $\zeta^{-1}(\rho_L)$ is close to the largest intensity found in an output, the bound is tight.

The notation for graphs in this section is as follows. The server intensities are shown by the bar heights, whereas the average intensity of servicing instances and the inverse of the bound are shown by the solid and dotted lines, respectively. Intensity values are shown along the y–axis. Server labels are shown along the x-axis.

In Figure 2 we observe the outputs using these kinds of curves. The set of three plots in Figures 2(a), 2(d), and 2(g)

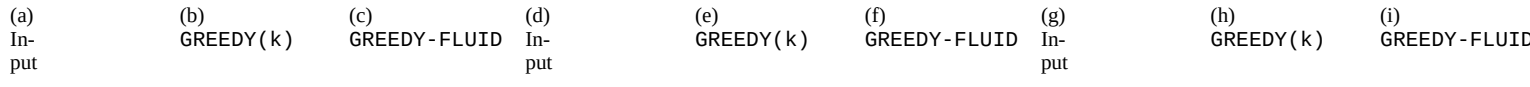

Fig. 2. Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian, fast–decaying, and linear inputs, $f_i = .1$, $1 \leq i \leq m = 16$. Note: a re-labeling of servers is done during both procedures.

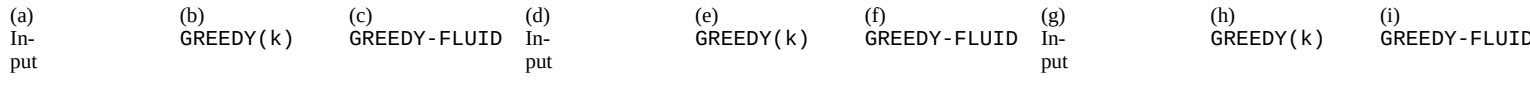

Fig. 3. Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian, fast–decaying, and linear inputs (all randomized), $f_i = .1$, $1 \leq i \leq m = 16$. Note: a re-labeling of servers is done during both procedures.

Fig. 4. Results obtained using GREEDY(k) and GREEDY-FLUID for fast–decaying input function, $f = .2$ in (a), (b), (c), $f = .3$ in (d), (e), (f), $m = 16$. Note: a re-labeling of servers is done during both procedures.

depict each of the three inputs as described. Each of these configurations correspond to not applying load–sharing, but having a static provisioning instead. In Figures 2(b) and 2(c), we show the results after algorithms GREEDY(k) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across a gaussian curve. In Figures 2(e) and 2(f), we show the results after algorithms GREEDY(k) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across the fast–decaying curve as described. In Figures 2(h) and 2(i), we show the results after algorithms GREEDY(k) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across a line also as described. In all cases, $f_i = .1$, for all $a_i \in \mathcal{A}$. We observe that in all outputs the maximum intensity is reduced to approximately the average value, hence indicating to be close to an optimal value. The largest intensities found in the outputs (of GREEDY(k) of Figures 2(b), 2(e), 2(h) are a reduction to about 63%, 50%, and 54% of the largest intensities in each respective inputs. The same applies for the outputs of the GREEDY-FLUID algorithm in Figures 2(c), 2(f), 2(i). In the case of the outputs from GREEDY-FLUID the bounds are close to the largest intensity obtained in the outputs as expected.

Another type of input is obtained from selecting m random numbers between 0 and m, typically a sorted sequence $\{x_i\}$, $x_i = U(0, m)$, $1 \leq i \leq m$, U a generator that outputs uniformly distributed numbers between 0 and m and applying the functions described previously to each of the x values in the sequence $\{x_i\}$ As a consequence, the read intensities of the service's instances are also randomized.

Figure 3 depics results obtained when randomizing the inputs as described. We show in Figures 3(a), 3(d), 3(g) a set of three plots which depict the inputs for the randomized gaussian, fast–decaying and linear–increasing inputs, respectively. In Figures 3(b) and 3(c) the largest intensity in both outputs (of GREEDY(k) and GREEDY-FLUID, respectively) is reduced to about 63% of the largest intensity in the inputs. In the case of outputs (of GREEDY(k) and GREEDY-FLUID, respectively) shown in Figures 3(e), 3(f), for the fast–decaying curve, and also outputs (of GREEDY(k) and GREEDY-FLUID, respectively) shown in Figures 3(h) and 3(i), similar comparisons indicate a reduction of the largest intensities in each of the cases to about 75% and 58%, respectively, of the largest intensity in the respective inputs.

Next we observe in Figure 4 inputs and sets of results for the cases where $f_i = .2$ in Figures 4(a), 4(b), and 4(c) and $f_i = .3$ in Figures 4(d), 4(e), and 4(f), $1 \leq i \leq m$, for the fast-decaying curve as input. In this case the values of f_i are higher than those in Figure 2. In Figure 4(b) the largest intensity output from GREEDY(k) is 62% of the largest intensity in the input. For the output from GREEDY-FLUID in Figure 4(c) the largest intensity is 54% of the largest intensity in the input. In Figure 4(e) the largest intensity output from GREEDY(k) is 67% of the largest intensity in the input, whereas for GREEDY-FLUID (Figure 4(f)) it is 57%. This results show a more significant difference between the outputs of GREEDY(k) and GREEDY-FLUID as $f = f_i$ increases. Here, the bound for GREEDY(k) is reasonably close to the largest intensity found in the output, but for GREEDY-FLUID the bound is worse, which is a result of having v_{max} larger as result of values of f_i, $1 \leq i \leq m$, also larger.

We also study inputs and outputs of fast-decaying and gaussian inputs where f_i is not equal for all servers but a decreasing linear function of the server label, described by $f_i = 0.2(1 - \frac{i}{m})$. Results are shown in Figure 5, where Figure 5(a)

(a) Input

(b) GREEDY(k)

(c) GREEDY-FLUID

Fig. 5. Results obtained using GREEDY(k) and GREEDY-FLUID for linear input, f described by a linear, decreasing function, $m = 16$. Note: a re-labeling of servers is done during both procedures.

(a) Input

(b) GREEDY(k)

(c) GREEDY-FLUID

Fig. 6. Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian input, write intensities $f_i\gamma_i$ equal for any $a_i \in \mathcal{A}$, $m = 16$. Note: a re-labeling of servers is done during both procedures.

depicts the input and Figures 5(b) and 5(c) depict the outputs for GREEDY(k) and GREEDY-FLUID, respectively. Here, $\bar{f} = .07$,an average weighted on the input demands, and the input demands are given by the linear increasing curve. The outputs exhibit largest intensities which are about 54 % of the largest intensity in the input for Figure 5. The bounds indicate values close to the ones found by the largest intensity in each of the outputs.

Finally we study cases in which the writing intensity is equal for all instances, i.e. $v_i = f_i\gamma_i = v_j = f_j\gamma_j$ for any $a_i, a_j \in \mathcal{A}$. Therefore, in this case the fragmenting cost of an instance's read intensity is a constant, regardless of the instance. We show in Figure 6 an example of input given by a gaussian curve and equal write intensities. Here, Figure 6(a) depicts the input, whereas Figures 6(b) and 6(c) plot the output for GREEDY(k) and GREEDY-FLUID, respectively. The outputs of both GREEDY(k) and GREEDY-FLUID show a reduction to about 63 % of the largest intensity in the output from the largest intensity from the input. The bound gives a value close to the one found by the largest intensity, especially in the case of GREEDY-FLUID in Figure 6(c).

V. Fast fluctuation of demands: preliminary model and analysis

In the previous section we examine simple algorithms that can be applied to distribute servicing across a set of servers and to reduce intensity and consequently response time. It is then assumed that averages of instances' intensities over measurement intervals give an accurate description of the system. These averages are snapshots taken over measurement intervals. But in some systems demands can fluctuate at a rate larger than the rate at which measurements are taken. Furthermore, the measurement rate can be constrained by practical limitations, thus cannot be made as small as desired. Examples include any instance of flash demand, i.e., demands that peak up unexpectedly and go down in a small timescale. For instance, activity of an auctioning system can have sudden, short-lived peaks due to a submitted bid that triggers a series of other bids from other users competing in the same auction. It is also well known that auctions' activity increases when close to their deadlines for submitting bids.

It could happen in a scenario of two instances a_1 and a_2 that a_1 exhibits high demand, a_2 exhibits low demand, and vice–versa, even if in the long–run both exhibit equal average intensity. The key idea here is to explore the possibility of statistical multiplexing across the demands of the two servers by having load sharing of both instances by servers s_1 and s_2. In this case a system with servers containing equal average response times can differ greatly with fluctuation of demand.

The model is extended to describe the intensity process for instances a_1 and a_2 by modulated processes each that describes demands in high and low demand periods. For an instance a_i read intensities in high and low demand periods are γ_i and $\gamma_i^{'}$, respectively, and for write intensities in high and low demand periods v_i and $v_i^{'}$, respectively, $i = \{1, 2\}$. The total intensity for an instance a_i in high and low demand periods is ω_i and $\omega_i^{'}$, respectively, which correspond simply to the sum of read and write intensities, i.e., $\omega_i = \gamma_i + v_i$ in high–demand periods and $\omega^{'} = \gamma_i^{'} + v_i^{'}$ in low–demand periods.The instance's intensity is high with probability p_i, and, conversely, low with probability $1-p_i$. Similar to the description in Section III, the ratio between the write intensity v_i and the read intensity γ_i is $f_i = \frac{v_i}{\gamma_i} = \frac{v_i^{'}}{\gamma_i^{'}}$.

A single server is described by a processor sharing queue with Poisson arrivals and capacity c.

In particular we consider the case in which $\omega_1 = \omega_2 = \omega$, $\omega_1^{'} = \omega_2^{'} = \omega^{'}$, $\gamma_1 = \gamma_2 = \gamma$, $\gamma_1^{'} = \gamma_2^{'} = \gamma^{'}$, $v_1 = v_2 = v$, $v_1^{'} = v_2^{'} = v^{'}$, and $p_1 = p_2 = p$. This case is interesting since it yields equal averages for both servers. Since in a configuration given by static provisioning the response times are already balanced, one might think by a simple intuition that the consistency cost increases response time when applying load sharing, but we demonstrate that this is not necessarily the case. In fact, we proceed to demonstrate in this section that applying load sharing can reduce response times.

The intensity $\rho(s)$ at a server is a random variable that is a function of the number of instances placed at s and the state (high, low) for each of the instances. It is convenient to define random variables N_1 and N_2, each describing if instances a_1 and a_2, respectively, are in high demand periods

($N_1 = 1$, $N_2 = 1$) or in low demand periods ($N_1 = 0$, $N_2 = 0$). It is further convenient to define the joint probability $\pi_{u_1,u_2} = P(N_1 = u_1, N_2 = u_2) = p^{u_1+u_2}(1-p)^{2-u_1+u_2}$. The probability that a random request is one of instance a_1 is $\beta_1 = \frac{u_1\omega+(1-u_1)\omega^{'}}{(u_1+u_2)\omega+(2-u_1-u_2)\omega^{'}}$, conditioned on $N_1 = u_1$ and $N_2 = u_2$. Similarly, the probability that a random request is one of instance a_2 is $\beta_2 = \frac{u_2\omega+(1-u_2)\omega^{'}}{(u_1+u_2)\omega+(2-u_1-u_2)\omega^{'}}$, conditioned on $N_1 = u_1$ and $N_2 = u_2$. The average response times for a unit–size job conditioned in the states of instances a_1 and a_2 are, respectively, $E[T|N_1 = u_1] = \frac{1}{c-(u_1\omega+(1-u_1)\omega^{'})}$ and $E[T|N_2 = u_2] = \frac{1}{c-(u_2\omega+(1-u_2)\omega^{'})}$. The static provisioning configuration yields an average response time as follows.

$$E[T]_{z=1} = \sum_{u_1=0}^{1}\sum_{u_2=0}^{1}\beta_1 E[T|N_1 = u_1]\pi_{u_1,u_2} + \sum_{u_1=0}^{1}\sum_{u_2=0}^{1}\beta_2 E[T|N_2 = u_2]\pi_{u_1,u_2}, \quad (12)$$

where $z = 1$ says that each of the instances is served at only one server, hence the static provisioning approach. For load sharing the response time conditioned on both state variables $N_1 = u_1$ and $N_2 = u_2$ is $E[T|N_1 = u_1, N_2 = u_2] = \frac{1}{c-\left((u_1+u_2)\omega+(1-u_1)(1-u_2)\omega^{'}\right)\left(1-\frac{1}{2(1+f)}\right)}$. When f is small, say $f = 0$ for the extreme case, and both instances are in a high–demand period, i.e., both $N_1 = u_1 = 1$ and $N_2 = u_2 = 1$, the intensity is equal to the intensity in static provisioning. If only one of them is in high–demand period, however, then the intensity can be cut by a half from the factor $1 - \frac{1}{2(1+f)}$, which can result in significant reduction in response time. The reduction in average also occurs since the probability of having a single instance at high–demand period is higher than the probability of having both at high–demand periods for p small. Finally, the average response time using load sharing is derived:

$$E[T]_{z=2} = \sum_{u_1=0}^{1}\sum_{u_2=0}^{1} E[T|N_1 = u_1, N_2 = u_2]\pi_{u_1,u_2}, \quad (13)$$

where $z = 2$ says that each of the instances is served on both servers, hence the load sharing approach.

The graphs in Figure 7 show the response time along the y–axis, whereas the fraction f of read requests in Figure 7(a) and the probability p are shown along the x–axis in Figure 7(b). The scenario here is given by $\omega = 1$ in high periods, $\omega^{'} = .1$ in low periods, $c = 2.5$, $f = .111$ (Figure 7(b)), and $p = .2$ (Figure 7(a)).

In Figure 7(a) the curves show the results obtained for $z = 1$, and $z = 2$, meaning that only one ($z = 1$) instance serviced per server (static provisioning) and two instances ($z = 2$) serviced at each server (load sharing). The results when varying f indicate the expected tradeoff in the amount of read requests, i.e., starting from approximately $f = 0.8$ downward the response times using $z = 2$ (load sharing) are smaller than $z = 1$ (static provisioning).

In Figure 7(b), by observing the probability p of an instance being on a high–demand period such that $p \leq 0.8$, the response times when $z = 2$ are smaller than the ones when $z = 1$. When p is high, in this case $p > 0.8$, both instances are very likely to be busy simultaneously, hence load sharing does not introduce a better result than a static provisioning solution. By contrast, when $p \leq 0.8$, the probability that either one of the instances (but not both) being in high–demand period and the other one in a low–demand period is larger than the probability that both are on high–demand period. This fact, along with the better load distribution, explains why load sharing results in smaller average response times. The exception happens when p is low, since most likely both are on low–demand period and the probability of either one being on high–demand becomes smaller. This fact explains why in Figure 7(b) as p decreases from $p = 0.4$ downward, the gap between the two curves gets narrower.

(a) Response times as function of ratio between read and write intensities.

(b) Response times as function of probability p of high demands.

Fig. 7. The case of two servers and two instances. Here, $\omega = 1$ in high periods, $\omega^{'} = .1$ in low periods, $c = 2.5$, $f = .111$ (in Figure 7(b)), and $p = .2$ (in Figure 7(a)).

VI. Conclusion

In this work we have studied the impact of load sharing in provisioning services with consistency requirements. A provider must take into account that consistency requirements place an extra burden to servers that "share" load. We model such servicing systems to investigate the fundamental factors on how to

apply load sharing.

The problem of minimizing the maximum intensity across servers as a way to reduce response times is found to be NP–hard. We have defined greedy algorithms for this problem and analyzed the outcomes from these greedy algorithms. The analysis shows that for the outcome of the `GREEDY-FLUID` algorithm, the largest intensity is always within a constant factor of the optimal. We observe that by applying load–sharing response times can be significantly reduced, since in quantitative analysis of representative cases the largest intensity can drop to approximately 50% of the largest intensity in a configuration in which all service instances are each served by a single server.

We also demonstrate that, even when response times given as result of static provisioning are equal in all servers, average response times when applying load sharing are smaller, if servicing instances fluctuation of demands are asynchronous. We show, for example, a simple scenario, in which intensities of two servers are equal with each of them serving a single instance, and by having the two servers serving both instances, response times are made approximately 10% smaller. This result is insightful and we can definitely expect a greater reduction of response times for comparisons of results that consider load–sharing to those that do not for larger number of instances and larger number of servers.