Utility-maximizing Server Selection

Truong Khoa Phan, David Griffin, Elisa Maini, Miguel Rio
University College London, UK
Email: {t.phan, d.griffin, e.maini, miguel.rio}@ucl.ac.uk

Abstract—**This paper presents a new method for selection between replicated servers distributed over a wide area, allowing application and network providers to trade-off costs with quality-of-service for their users. First, we create a novel utility framework that factors in quality of service metrics. Then we design a polynomial optimization algorithm to allocate user service requests to servers based on the utility while satisfying transit cost constraint. We then describe an efficient - low overhead distributed model with the need to only know a small subset of the data required by a global optimization formulation. Extensive simulations show that our method is scalable and leads to higher user utility compared with mapping user requests to the closest service replica.**

I. Introduction and Motivation

As the Internet becomes the enabler for more types of services with a wider spectrum of requirements, pressure is being put onto the Internet ecosystem to facilitate service placement and to select the best replica for each user request at each instant in time. This replication always involves multi-stakeholder trade-offs involving costs (deployment and traffic related) and user quality of service (QoS).

There are many drivers for service replication, including server resilience, network diversity, and proximity of servers to users. Deploying services closer to the users allows the application providers to improve on QoS metrics like latency and/or throughput for all users. Some frameworks, like fog computing [1], even attempt to put service instances at the extreme edge of the network in locations such as access points.

Although in theory this replication could be optimal, in practice there are several obstacles: deployment costs vary between geographical areas and may be prohibitive in some locations, demand forecasting is inaccurate, flash crowds are unpredictable. Efficient allocation of user requests to service replicas will have to rely on a service selection at query time.

Service quality has two major sets of component metrics, relating to computation and networking parameters. Servers will have to be properly provisioned for the arrival rate and holding time of user requests otherwise users will be not served or blocked, increasing application latency. Network distance will have primarily an effect on end-to-end delay but, in many scenarios, causes an increase in packet loss and/or a decrease in good-put. The service selection system will have to take into account both computation and networking factors to optimize its selection.

Resolution involves converting a service name to a specific network locator for the selected replica. Our work assumes that the user's ISP is in the best position to make this selection. The ISP has accurate information regarding the user's position in the network, the current network status and, furthermore, it allows the ISP to apply traffic network policies in the selection process to reduce traffic costs. A centralized approach would, in theory, allow global optimization but it would often be unscalable and unrealistic. A central entity would not have access to information on the detailed user position, the network topology or current network status and would be incapable of implementing ISPs' specific traffic policies as it would have to arbitrate between conflicting policies of different ISPs which would be problematic from a business point of view. For those reasons, our server selection model can be implemented in a similar way of PaDIS [2] which allows ISPs to better assign users to servers by using their knowledge about network conditions and user locations.

ISBN 978-3-901882-83-8 © 2016 IFIP

In brief, the contributions of our work are as follows:

- Firstly, we introduce the *utility function* relating to one or more QoS metrics that allows application providers to define based on their application's requirements.
- Secondly, we design a *polynomial centralized optimization algorithm* that allows ISPs to redirect their users to the best replica, allowing to trade-off their traffic costs with users' QoS. In addition, the model allows to optimize for multi-services at the same time.
- Finally, we propose a *simple - efficient distributed model* that allows ISPs to run a local version of the selection algorithm without the need for global knowledge of all service replicas and network conditions.

This paper is organized as follows: Section II introduces the utility-maximizing server selection model and is followed by the optimization formulation in Section III. Section IV presents extensive evaluation. We finish by surveying related work in Section V and conclusions in Section VI.

II. Utility-maximizing Server Selection

The goal of utility-maximizing server selection is to provide the highest QoS for the greatest number of users. Our framework unifies the objectives of several stakeholders and the quality of service of the end users. In our approach, the stakeholders involved in service selection are as follows:

- Execution zones (EZ): These are the entities running the computational aspect of the distributed application. Typically they will be cloud/data centers but they can be smaller micro-data centers. They can be run by the ISPs themselves (current trends point to this being an importance new revenue stream for them). Whatever the scenario, they want to maximize their revenue.

- Application (Service) Providers: These represent the organizations that wish to run distributed applications in execution zones. They may instantiate their replicas directly or use a third-party orchestrator (e.g. cloud broker). They will trade-off the deployment costs with the quality-of-service of their users. After deployment they publish the utility function of the service so that resolution algorithms can maximize this utility. It is noted that the utility function can simply be general (no sensitive) latency requirements for applications as shown in Fig. 2.
- Internet Service Provider: These will implement resolution algorithms to resolve users queries, mapping service names to locators. They will also trade-off the final quality of service of their users with the traffic costs imputed to them by these choices.

Application providers deploy service instances in execution zones. These replicas register with a local resolver in the ISP to which they are connected and send periodic updates. These messages contain for each service:

- The utility function of the service as described in section II-B.
- The number of available *session slots*. A session slot is a unit of measurement representing how many users can be accommodated simultaneously in a given service instance without blocking.
- Servicing execution statistics regarding the total demand on the service instance and the distribution of service duration times.

We define routing epoch as the interval between the resolution system making resolution decisions. Regarding to *session slots* announcement, if sessions are long compared to the routing epoch then the EZs simply announce a snapshot of what is available. However if session durations are short then an announcement of instantaneous availability is more-or-less meaningless. For example: assume a routing epoch of 10 seconds and a service S_1 with an average duration of 100 seconds and S_2 with an average duration of 1 second, and a single service instance for each service can each handle 2 sessions simultaneously. The EZ would announce 2 available session slots for S_1 as the current session is likely to last much longer than the routing epoch. However for S_2, if 2 available session slots are announced, it would mean that only two requests should be forwarded to that EZ, even though the currently active session (as well as those arriving in the near future) is very likely to end during the epoch. Therefore the number of session slot would be announced up to 20 for S_2, depending on the service arrival time.

Given those aforementioned stakeholders involving in the system, we present next the main criteria to do server selection.

A. Motivation Example

As an example in Fig. 1, assuming that there are two users requiring a voice service which is available in both EZ_1 and EZ_2. However, each EZ can serve only one user at a time or they announce only one available session slot. Latencies between users and EZs are shown in Fig. 1. For the voice service, we observe that if bandwidth is enough, and when the latency is equal or less than 20 ms, humans' ears cannot distinguish between audio and real speech or in other words, people do not feel any disturbance [3]. Therefore, 5 ms or 20 ms latency gives the same (and the best) QoS for voice services. Note that we consider here the latency for on-going services, not including setup time. As shown in Fig. 1, the classical closest based selection algorithm would have solution: (user 1 - EZ_1) and (user 2 - EZ_2) (dash lines) as the minimum total latency is $(5+30) = 35$ ms. It means that user 1 can have the best QoS while user 2 sees some disruption in the voice quality. However, we see a better solution should be (user 1 - EZ_2) and (user 2 - EZ_1) in which both users get the best QoS with 20 ms latency. This is the motivation of our work to define a utility function applying in the server selection problem.

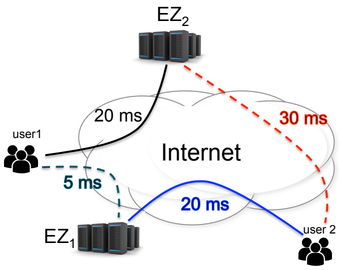


Fig. 1: Utility based vs. closest based selection

.

B. Utility Function

Our general utility function is grounded on practical research on quality of service utility [4], [5] and years of investigation on Mean Opinion Scores [6]. Our interval data points map to user ratings of *excellent*, *good*, *fair*, *poor* and *no service or blocked* (Fig. 2).

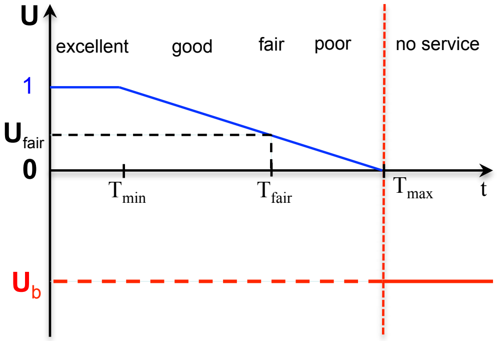


Fig. 2: Utility function vs. latency

.

In our utility framework, application providers determine utility function by the two latency thresholds: T_{min} and T_{max}.

Note that the utility is not restricted to only latency. In future work, we will extend the utility to be a combination of any QoS metrics such as latency, bandwidth, loss, etc. As shown in Fig. 2, we use a non-increasing piecewise linear utility function that is characterized by:

- If $t \leq T_{min}$: depending on the service type, an appropriate value of T_{min} is selected meaning that even if the latency reduces below this value, the improvement is not perceived by the users of that service, thus the utility is unchanged ($U_{max} = 1$). For instance, *voice over IP* requires $T_{min} = 20$ ms [3]; for *simple web services*, $T_{min} = 100$ ms gives users the feeling of instantaneous response [7].
- If $T_{min} < t \leq T_{max}$: QoS is within an acceptable range ($0 \leq U < 1$). User satisfaction reduces as the latency increases. We also define $T_{fair} \in [T_{min}, T_{max}]$ as the point from which users start to feel disappointed about the services as QoS is getting poor. Note that the value of T_{fair} is set depending on services and does not change the slope of the utility graph.
- If $T_{max} < t$: the request is *blocked* (*no service*) because the latency is beyond the acceptable range. Details on blocked requests are presented in Section III-B2.

Based on this utility function, the utility-maximizing solution for the problem in Fig. 1 will be (user 1 - EZ_2) and (user 2 - EZ_1) because both users will get the maximum utility with $t = T_{min} = 20$ms.

III. Server Selection Optimization

In this section, we present a mathematical formulation for the server selection problem that ISP's resolvers use to dynamically resolve names to locators. In general, the goal is to direct user requests to suitable execution zones (EZs) and satisfy a predefined objective function. We use the utility function defined in Section II-B to measure user satisfaction. We use linear programming to formulate the server selection problem which maximizes the total utility of all users while taking into account constraints on the data transit cost.

A central pre-requisite for our model is the existence of a forecasting demand component that provides an input to the optimization algorithm. Although client demand varies with time, work in the literature [8] points to reasonably stable demand within 10 minutes intervals. Note that, we aggregate individual users with the same preference to form a group. This can be done according to users' postal codes [8] or by users' IP prefixes [9]. Aggregation of this kind reduces the quantity of input variables for the optimization and also stabilizes request rates per-group [8].

A. Problem description

- Input: estimated user requests ($\mathcal{D}$); two threshold values (T^{ij}_{min} and T^{ij}_{max}) for each pair of (user group i, service j) defined in the utility function (Fig. 2); latency between user group i and EZ z for a specific service j is l^j_{iz}; unit data transit cost (c_{iz}); the maximum budget ($COST$) and available session slots at each EZ (S^j_z).
- Objective: maximize the performance (total utility) of users for multi-services j while considering the trade-off between the performance and the data transit cost.
- Output: $x^j_{iz} \in [0, 1]$: fraction of user group i connecting to EZ z for service j.

We define a utility function as described in Section II-B as follows:

$$u_{ij} = \begin{cases} 1 & \text{if } t_{ij} \leq T^{ij}_{min} \\ \frac{-t_{ij}+T^{ij}_{max}}{(T^{ij}_{max}-T^{ij}_{min})} & \text{if } T^{ij}_{min} < t \leq T^{ij}_{max} \\ U_b & \text{otherwise} \end{cases}$$

The utility is defined for each pair of (user group i, service j). When the latency is larger than T^{ij}_{max}, the request is considered to be blocked. We set U_b to be a small negative value to indicate the utility of a blocked request. More details on how to set value for U_b are presented in Section III-B2. We first present a centralized model and then introduce a distributed one which is more suitable for Internet-scale deployment.

B. Centralized model

Given the key notations in Table I, we use linear programming to formulate the utility-maximizing server selection problem.

1) Linear Program Formulation:

$$\max[\sum_{(i,j)\in\mathcal{D}} u_{ij}] \tag{1}$$

s.t.

$$\sum_{z\in\mathcal{Z}} x^j_{iz} = 1 \qquad \forall(i,j) \in \mathcal{D} \tag{2}$$

$$\sum_{i\in\mathcal{I}} d_{ij} x^j_{iz} \leq S^j_z \qquad \forall j \in \mathcal{J}, z \in \mathcal{Z} \tag{3}$$

$$t_{ij} = \sum_{z\in\mathcal{Z}} l^j_{iz} x^j_{iz} \qquad \forall(i,j) \in \mathcal{D} \tag{4}$$

$$y_{ij} \geq 0 \qquad \forall(i,j) \in \mathcal{D} \tag{5}$$

$$y_{ij} \geq t_{ij} - T^{ij}_{min} \qquad \forall(i,j) \in \mathcal{D} \tag{6}$$

$$u_{ij} = \frac{T^{ij}_{max} - T^{ij}_{min} - y_{ij}}{T^{ij}_{max} - T^{ij}_{min}} \qquad \forall(i,j) \in \mathcal{D} \tag{7}$$

$$\sum_{z\in\mathcal{Z}} \sum_{(i,j)\in\mathcal{D}} c_{iz} b_{ij} x^j_{iz} \leq COST \tag{8}$$

$$x^j_{iz} \in [0, 1], u_{ij} \leq 1 \qquad \forall(i,j) \in \mathcal{D}, z \in \mathcal{Z} \tag{9}$$

Explanation:

- The objective function (1) maximizes the total utility over all user groups.
- Constraint (2): all the requests of user group i for a specific service j have to be served.
- Constraint (3): each EZ has a limited capacity (bandwidth and computational resources) for deploying instances of a specific service type. It may be the case that some special services (e.g. those that require GPU processing) can only be deployed at certain EZs with the necessary hardware

TABLE I: Key Notations (in Alphabetical Order)

b_{ij}	bandwidth required by user i to get service j
$COST$	the maximum transit cost (budget)
c_{iz}	unit transit cost between user i and EZ z
$\mathcal{D}$	set of user requests $\mathcal{D} = \{(i, j), \forall i \in \mathcal{I}, j \in \mathcal{J}\}$
d_{ij}	requested session slot of user i to service j
$\mathcal{I}$	set of user groups $\mathcal{I} = \{i\}$
$\mathcal{J}$	set of services $\mathcal{J} = \{j\}$
l^j_{iz}	latency between user i and EZ z for service j
S^j_z	available session slot of service j at EZ z
t_{ij}	average latency of user i to get service j
$\mathcal{Z}$	set of execution zones (EZ) $\mathcal{Z} = \{z\}$
U_b	utility value of a blocked user
u_{ij}	utility of user i when getting service j
x^j_{iz}	fraction of user i connects to EZ z to get j
y_{ij}	variable used to compute the utility

available. This constraint ensures that the number of session slots available in an EZ is sufficient to serve user requests.

- Equation (4) is used to compute the average latency for the user group i to get the service j. We model connectivity as a full mesh between user groups and EZs. However, for the input of the LP, we do not consider any connections between user group i and EZ z to get service j if the latency $l^j_{iz} > T^{ij}_{max}$. This step guarantees that, even without adding explicit constraints in the model, the latency for any user group i to connect to any EZ z to get service j is less or equal to T^{ij}_{max}.
- Constraint (5) - (6) ensure that $y_{ij} \geq 0$ if $t_{ij} \leq T^{ij}_{min}$, otherwise $y_{ij} \geq t_{ij} - T^{ij}_{min} > 0$.
- Equation (7) is used to model the utility function defined in Section III-A. There are two possibilities:
  - If $t_{ij} \leq T^{ij}_{min}$, based on constraints (5) - (6), y_{ij} can be any value greater or equal to 0, however, due to the objective function maximizing utility (7) the formulation will choose a minimum value of y_{ij}, or in other words, y_{ij} is set to 0 and thus, $u_{ij} = 1$.
  - Similarly, if $t_{ij} > T^{ij}_{min}$, the formulation will choose $y_{ij} = t_{ij} - T^{ij}_{min}$ and thus $u_{ij} = \frac{-t_{ij}+T^{ij}_{max}}{T^{ij}_{max}-T^{ij}_{min}}$.
- Constraint (8) limits the data transit cost. As shown in [9], [10], the linear transit cost we use here is also a good approximation for the 95-th percentile transit cost.

An important remark is that our formulation allows to optimize for multi-services at the same time, i.e. maximizing the total utility for all pairs of (user i, service j). For instance, assume that we have video streaming and voice services, each potentially has different QoS requirements. If we optimize for video streaming first, then the remaining resources are dedicated for the voice service and vice versa. This will end up with a sub-optimal solution. Our model allows to optimize for multi-services at the same time, therefore it guarantees to find a global optimal solution.

The optimization formulation above is a (pure) linear programming model one; hence it can be solved efficiently in polynomial time. The number of variables x^j_{iz} in the LP problem is $|\mathcal{I}| \times |\mathcal{Z}| \times |\mathcal{J}|$ where $|\mathcal{I}|$ is the number of user groups, $|\mathcal{Z}|$ is the number of EZs and $|\mathcal{J}|$ is the number of service types. Since $|\mathcal{Z}|$ and $|\mathcal{J}|$ are usually much smaller than $|\mathcal{I}|$, the worst case complexity of the LP problem will be $O(|\mathcal{I}|^{3.5})$ [9]. We report the execution time of the algorithm in Section IV-C1.

2) Blocked user requests: When there are not enough resources (session slot or cost) the constraints (3) and (8) in the LP are violated and the LP ends up with no solution. We address this by allowing user requests to be *blocked* when there are insufficient resources. To model this, we define a virtual EZ with very large capacity so that the constraint (3) cannot be violated. The transit cost between users and the virtual EZ is zero. The latency between all users to this virtual EZ is set at a value which is larger than T_{max}, therefore the utility for a blocked request is $U_b < 0$ (Fig. 2). We evaluate different values of U_b and show its impact on the number of requests to be blocked when running the optimization model (Section IV-A2). Intuitively, the closer to 0 the value of U_b is, the more possibility for requests to be blocked because there is a reduced difference in utility between a request at T_{max} ($U_{T_{max}} = 0$) and a blocked one (U_b). It is noted that, with the virtual EZ, the LP always finds a feasible solution because the constraints (3) and (8) cannot be violated, but the total utility could be extremely (negative) small. And those requests that have to go to the virtual EZ are considered to be blocked.

C. Distributed model

Although our centralized optimization model can be solved in polynomial time, it is impractical in real deployments as a single global resolver would be required to collect information from all EZs and networks and also to handle resolution requests from all users. Moreover we want to put the ISP (*resolver*) at the center of decision making in order for local traffic policies to be applied.

Designing an efficient distributed algorithm is a classical problem [8], [10], [11], and it would satisfy the following general requirements:

- *(1) Low overhead:* small number of control messages exchanged. In addition, it should guarantee a fair share based on demand of each resolver.
- *(2) Convergence:* the algorithm is guaranteed to be always converged to a stable solution.
- *(3) Efficiency:* solution of the distributed algorithm is close to the centralized one.

Existing work in literature can satisfy the requirements (2) and (3) by using optimization decomposition methods [8], subgradient methods [11] or alternating direction method of multipliers [10]. However, they end up with high complexity formulation and require high control overhead. Potentially, control messages can be exchanged between (in both directions): resolvers - resolvers, resolvers - EZs, and EZs - EZs. In this work, we propose a novel distributed model satisfying all the three aforementioned requirements. Compared with existing work, our model is simpler (still can be solved

TABLE II: Key Notations in Distributed Algorithm

$A_i^z(k)$	allocated session slots at z by R_i in epoch k
C^z	total session slots at EZ z
k	epoch (iteration) number
M	number of resolvers that share an EZ
N	number of EZs that one resolver can see
R_i	resolver i
$S_i^z(k)$	available session slots at z seen by R_i in epoch k
Z	set of execution zones

in polynomial time) and low overhead in which only one-way control messages from EZs to resolvers are needed. In addition, the messages exchanged are simple as we describe later.

1) Distributed algorithm: We divide the time into **intervals** in which we assume the traffic demand is unchanged (e.g. 10 minutes as observed in [8]). Each interval is sub-divided into **epochs** and the distributed algorithm is run at the beginning of each epoch. We call *visibility set* be a subset of EZs that are closest to a resolver and can be seen by that resolver.

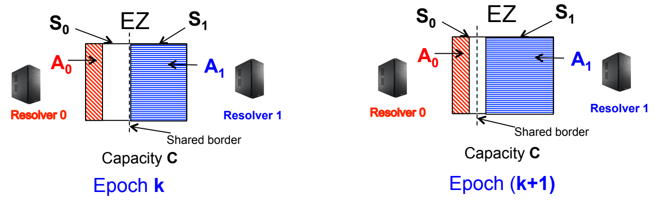


Fig. 3: EZ z is shared by two resolvers R_0 and R_1

.

We introduce some notations used in the distributed algorithm (Table II). Considering an EZ z with total available session slots C^z which is shared by M resolvers. At an epoch $k \geq 0$, let a resolver $R_i (0 \leq i \leq M-1)$ see $S_i^z(k) \leq C^z$ session slots from the shared EZ. To guarantee capacity constraint, we have $\sum_{i=0}^{M-1} S_i^z(k) \leq C^z$. Let $A_i^z(k) \leq S_i^z(k)$ be the number of session slots that the resolver R_i allocates for its users to connect to the EZ z at the epoch k. A visualization of those notations are shown in Fig. 3. The algorithm, step-by-step, at each resolver is as follows:

1) *At the beginning of each interval*: collect the latest estimated local user requests and network metrics (e.g. latency between users and EZs).
2) *At the beginning of each epoch*: each EZ announces the latest capacity (C^z) and the total-in-allocation (total-in-use) session slots ($\sum_{i=0}^{M-1} A_i^z(k)$) at that EZ to resolvers.
3) Each resolver updates available session slots that it can use in the next epoch as follows:
$$S_i^z(k+1) = A_i^z(k)\big[1 + \frac{C^z - \sum_{i=0}^{M-1} A_i^z(k)}{\sum_{i=0}^{M-1} A_i^z(k)}\big] \quad (10)$$
if $\sum_{i=0}^{M-1} A_i^z(k) = 0$, we set $S_i^z(k+1) = C^z$.
4) Given new available session slots from EZs, resolvers execute the linear program in Section III-B to find which users should connect to which EZs to get services.

By using the equation (10), we show that the distributed algorithm satisfies all the requirements mentioned in III-C:

- Each resolver requires local user demand and the session slots have been used in the previous epoch ($A_i^z(k)$). In addition, each resolver uses the LP in III-B, thus the distributed algorithm can be solved in polynomial time.
- Low overhead: only one-way message from EZs to resolvers to update information about total capacity (C^z) and total in-use session slots at the previous epoch ($\sum_{i=0}^{M-1} A_i^z(k)$). In addition, we show that the equation (10) also achieves the *fair share on demand* requirement. As shown in Fig. 3, at epoch k, the resolver R_0 just uses a small fraction of its shared available session slots ($A_0 < S_0$) while R_1 requires all the slots that it can see ($A_1 = S_1$). Therefore, in the epoch $(k+1)$, we should move the shared border to the left (but do not touch the red area - the allocated slots of R_0) so that there will be more free space for R_1 to forward its requests to the EZ if needed. This can be done automatically by using the equation (10) (see the example in III-C2).
- We show, both by mathematical proof and simulations that local decisions always converge within a handful of iterations and the solution of distributed algorithm is close to the centralized one. However, because of limited space, we omit the mathematical proof in this paper and will present it in a journal version.

Initially, when services are first deployed, each EZ announces its available session slots to all resolvers that can see it. Given the available session slots and the local user demand, each resolver executes the linear formulation in section III-B to find a solution for its users. In this initial step, EZs can be overloaded as they are shared by many resolvers, but there is no message between resolvers to say that. However, by using the equation (10) to update available capacity at EZs after each epoch, the capacity constraints are not violated after the initial step. We present a simple example to make the algorithm clear.

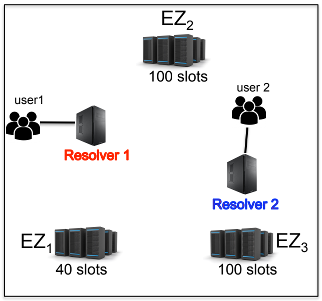


Fig. 4: Example of distributed algorithm

.

2) Examples of distributed algorithm: Assume that user 1 requires 100 slots and user 2 requires 80 slots. Capacities of EZs are shown in Fig. 4. The latencies between resolvers, users and EZs are as follows:

- $l(R_1, EZ_2) < l(R_1, EZ_1) < l(R_1, EZ_3)$
- $l(R_2, EZ_3) < l(R_2, EZ_2) < l(R_2, EZ_1)$

- $T_{min} < l(usr_1, EZ_2) < l(usr_1, EZ_1) < l(usr_1, EZ_3)$
- $T_{min} < l(usr_2, EZ_2) < l(usr_2, EZ_3) < l(usr_2, EZ_1)$

Recall that depending on the size of the *visibility set*, we can have different solutions for the server selection problem. Using the above network metrics, we consider the example with two scenarios:

- Scenario 1 (*visibility set size is* 1): the resolver 1 can only see EZ_2 (as EZ_2 is the closest EZ of R_1) and the resolver 2 can only see EZ_3. Therefore, the solution will be: the resolver 1 sends all 100 requests to EZ_2 and similarly, all requests of the resolver 2 go to EZ_3. This solution does not change if the user requests are unchanged between epochs.

- Scenario 2 (*visibility set size is* 2): resolver 1 can see (EZ_2 and EZ_1) and resolver 2 can see (EZ_3, EZ_2). Assume that the requests do not change, we present results for each resolver within 2 epochs (or 2 iterations of the distributed algorithm).

- Epoch 0:
  - Resolver 1 sees from EZ_1: $S_1^1(0) = 40$, and from EZ_2: $S_1^2(0) = 100$. As $l(usr_1, EZ_2) < l(usr_1, EZ_1)$, it forwards all 100 requests to EZ_2.
  - Resolver 2 sees from EZ_2: $S_2^2(0) = 100$, and from EZ_3: $S_2^3(0) = 100$. As $l(usr_2, EZ_2) < l(usr_2, EZ_3)$, it assigns all 80 requests to EZ_2.

  The total allocated session slots at EZ_2 is 180, and the EZ_2 is overloaded at epoch 0.
- Epoch 1:
  - Resolver 1 updates available session slots using the equation (10):
    - $S_1^1(1) = C^1 = 40$ (as $A_1^1(0) + A_2^1(0) = 0$)
    - $S_1^2(1) = 100 \times (1 + \frac{100-180}{180}) = 55$

  Solution after epoch 1 is: 40 slots go to EZ_1; 55 slots go to EZ_2 and 5 slots are blocked (as there are insufficient session slots).

  - Resolver 2 updates available session slots using the equation (10):
    - $S_2^3(1) = C^3 = 100$ (as $A_1^3(0) + A_2^3(0) = 0$)
    - $S_2^2(1) = 80 \times (1 + \frac{100-180}{180}) = 44$

  Solution after epoch 2 is: 44 slots go to EZ_2; 36 slots go to EZ_3 and no slots are blocked.

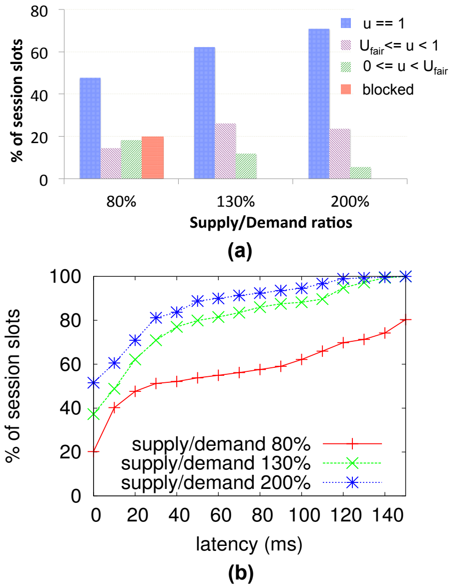
It is clear that, after epoch 1, thanks to the equation (10), no EZ is overloaded. In this example, the solution does not change after 2 epochs as long as the user demands do not change. It means that the distributed algorithm converges to a stable solution. On the other hand, session slots are assigned proportionally to the requirement of each resolver. For instance, in the stable solution, R_1 and R_2 respectively use 55 and 44 slots from the EZ_2. It is because in epoch 0, R_1 requires 100 slots while R_2 needs only 80 slots ($\frac{100}{80} \simeq \frac{55}{44}$). We call this as *fair share on demand*.

IV. Simulation Results

We solve the linear program model using IBM CPLEX solver [12]. All computations were carried out on a computer equipped with a 3 GHz CPU and 8 GB RAM. Our evaluation is through simulation and consists of 5 main parts. Firstly, we evaluate the algorithms with different parameters: $\frac{supply}{demand}$ ratios, U_b on blocking probability and visibility set sizes for the distributed algorithm. Next, we compare our novel utility-maximizing server selection (USS) with the classical closest based server selection algorithm. Then, we evaluate the distributed algorithm and compare with the centralized one. Next, we show the impacts of mismatch between supply and demand on the server selection solution. And lastly, we discuss on the inaccuracies of demand forecasting when using our algorithm.

We use a dataset with 2508 data centers distributed in 656 cities all over the world [13]. For the distributed model, we assume that each city has one resolver. Since data centers in a city are geographically close to each others, we group them as one execution zone (EZ). The capacity of an EZ is proportional to the number of data centers in that city. We assume that the services are available in all EZs. The data transit cost is based on the Amazon EC2 charging model. The user demand is modeled as Poisson process and is proportional to the population of each city [14]. The latency between users and execution zones are collected based on Haversine distance, the shortest distance between two points around the planet's surface [15].

A. Different Parameters for the Algorithm



(a)

(b)

Fig. 5: Different $\frac{supply}{demand}$ ratios

.

1) Different Supply/Demand Ratios: We first find server selection solutions for different $\frac{supply}{demand}$ ratios with the centralized algorithm. We set $T_{min} = 20$ ms, $T_{fair} = 100$ ms and $T_{max} = 150$ ms for all pairs of (group user, service). In Fig. 5, we show the utility and the Cumulative Distribution Function (CDF) of latency for three scenarios of $\frac{supply}{demand}$ ratio: 80%, 130% and 200%. $\frac{supply}{demand} = 80\%$ means that the total available capacity in all EZs is scaled down to equal to 80% of the total requests. As a result, 20% of the requests will be blocked while maximizing total utility of the served requests. In the CDF of latency in the scenario 80% (Fig. 5b), only 80% of requests receive service with less than $T_{max} = 150$ and the remaining requests are blocked. For the two other scenarios (130% and 200%), since there are sufficient session slots, no user request is blocked. Obviously, the greater the supply of session slots, the better solution we get in terms of utility and latency (Fig. 5).

2) Utility of a Blocked User: As shown in Section III-B2, our algorithm allows to block user requests while maximizing the total utility. By selecting different values of utility for a blocked request ($U_b < 0$), we obtain solutions with different blocking probabilities. We show in Fig. 6 the results for the *centralized algorithm* with different values of U_b. When U_b is close to 0, e.g. $U_b = -1.1$ or $U_b = -2$, blocking user requests does not incur much penalty in the total utility. A significant number of requests are blocked despite total utility being maximized. When U_b is much more smaller (e.g. $U_b = -100$), blocking a single request can dramatically reduce the total utility, thus the algorithm tries to avoid as many requests being blocked as possible. In Fig. 6, when $U_b = -100$, no user request is blocked.

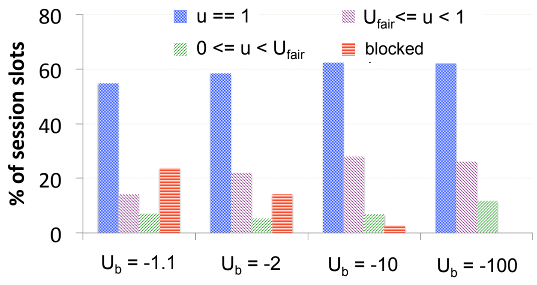


Fig. 6: Different values of U_b

.

In the remaining evaluation, if not stated otherwise, default values are used as follows: $\frac{supply}{demand}$ ratio = 130%, $U_b = -100$, $T_{min} = 20$ ms, $T_{fair} = 100$ ms and $T_{max} = 150$ ms.

3) Different visibility sets: In a distributed manner, each resolver only sees its local user demand and a subset of EZs which is called *visibility set*. We vary the size of the visibility set by changing the parameter N, the percentage of the total 656 execution zones that can be seen by a resolver. For example, $N = 0.3\%$ means that each resolver can see its 2 closest EZs.

Intuitively, when N increases, more session slots are available for a resolver to allocate user requests. As shown in Fig. 7, the percentage of blocked requests reduces as we increase N. However, as resolvers greedily allocate their user requests to the best EZs in their visibility set, users in "poor resource" areas do not have enough session slots and still we can see a few of users are blocked even $N = 100\%$. Note that in the centralized algorithm, there is no blocked user request when the $\frac{supply}{demand}$ ratio is 130%.

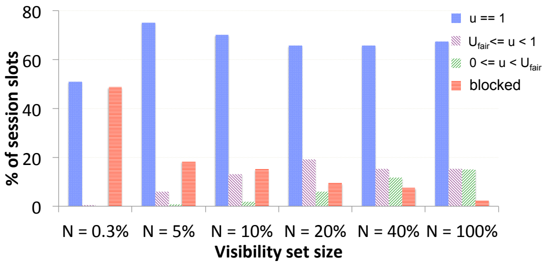
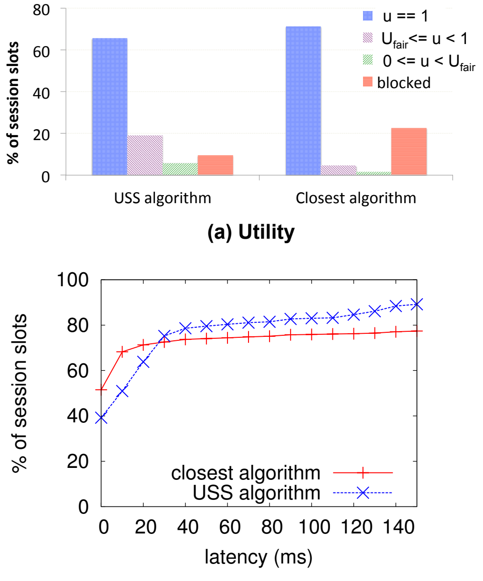


Fig. 7: Utility with different visibility set sizes

.

B. USS vs. closest selection algorithm



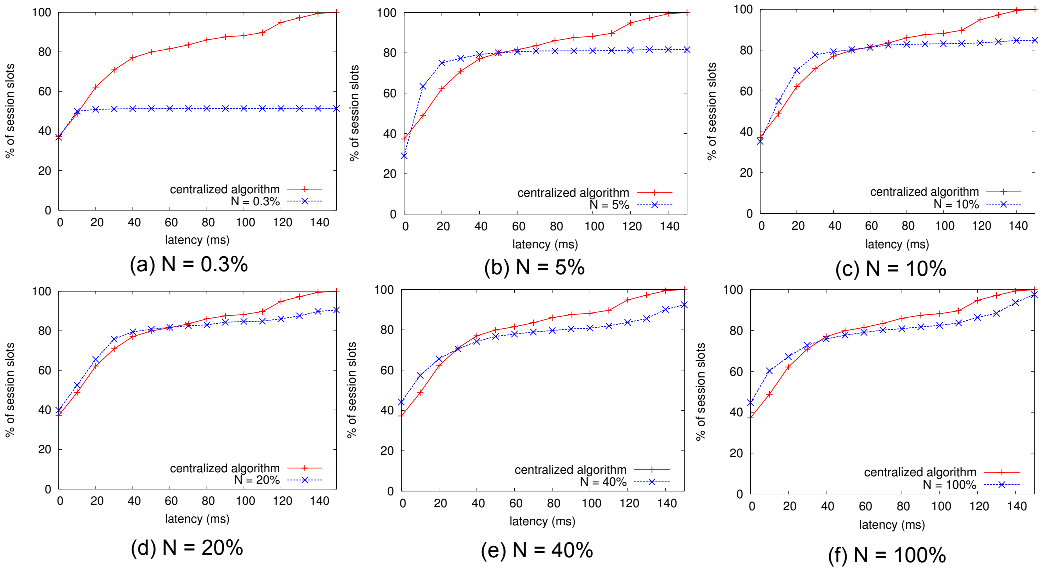
(a) Utility

(b) CDF latency

Fig. 8: Voice: USS vs. closest algorithm

.

Given the parameters in IV-A2, Fig. 8 shows a comparison between our utility-maximizing server selection (USS) and the classical closest algorithm with $N = 20\%$. The closest algorithm tries to allocate user requests to nearby EZs that have available session slots. If the closest EZ does not have available session slots, the algorithm considers the next closest one and so on. Only the case there is no available session slot within the latency range of T_{max}, requests have to be blocked. After finding the latency for user requests in the closest solutions, we compute the utility corresponding to the voice ($T_{min} = 20$ ms, $T_{fair} = 100$ ms and $T_{max} = 150$ ms [16]) (Fig. 8a). We can see the USS algorithm performs better with less blocking probability. This is because the USS algorithm is less greedy, providing more flexibility for requests to connect to many servers which have latency less than T_{min}. Taking a closer look at the CDF of latency (Fig. 8b), more requests get low latency in the closest algorithm, however more requests are also blocked due to its greedy behavior.



(a) N = 0.3%

(b) N = 5%

(c) N = 10%

(d) N = 20%

(e) N = 40%

(f) N = 100%

Fig. 9: Latency with different visibility set sizes

.

C. Distributed Algorithm

1) Distributed vs. centralized algorithm: We show in Fig. 9 the CDF of latency for the centralized and the distributed algorithms with different visibility set sizes. One of the goal when designing the distributed algorithm is that it should perform well, close to the centralized one. However, performance of the distributed algorithm depends on how much resource a resolver can see. As shown in Fig. 9, as visibility set size N increases, each resolver can see more available session slots, therefore less user requests are blocked. To report on execution time, the centralized algorithm with full knowledge of execution zones and user demands takes about 2 minutes to find an optimal solution, while the distributed algorithm only requires a few seconds to finish.

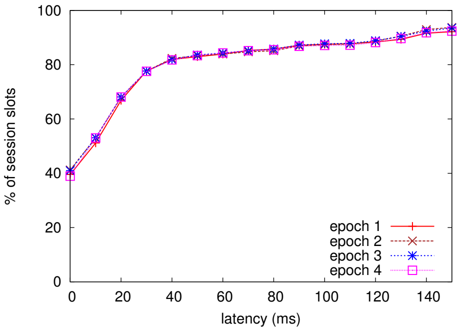


Fig. 10: Convergence of distributed algorithm

.

2) Convergence of distributed algorithm: To evaluate the convergence of the *distributed algorithm*, Fig. 10 shows the quality of the solution after 4 epochs in case visibility set size $N = 20\%$. We see that the four lines in Fig. 10 are nearly overlapped. After the first epoch, we already have a reasonably good solution and is close to the final solution. This result confirms a fast convergence of the distributed algorithm.

D. Mismatch between supply and demand

To evaluate the impact of mismatch between supply and demand, we first run the *centralized model* (section III) but without the capacity and the cost constraints. That is to find how many session slots are needed at every EZ for an optimal server selection solution. Then we scale these values to achieve 130% $\frac{supply}{demand}$ ratio. We call this configuration be the *perfect allocation*. Next, we create different levels of mismatch between supply and demand by varying a parameter "$X\%$ rand." (Fig. 11). This means that for each EZ, we remove $X\%$ of its session slots from the perfect allocation configuration. Then, we mix the removed session slots of all EZs and scatter them uniformly to all EZs. This guarantees that the total session slots of EZs in all cases (perfect allocation and "$X\%$ rand.") are the same. "0% rand." is equivalent to the perfect allocation while in "100% rand.", there is a uniform distribution of sessions slots between all EZs. Fig. 11 shows evaluation results for the *distributed algorithm* with visibility set size $N = 5\%$ with different value of "$X\%$ rand.". With the perfect allocation "0% rand.", the distributed algorithm

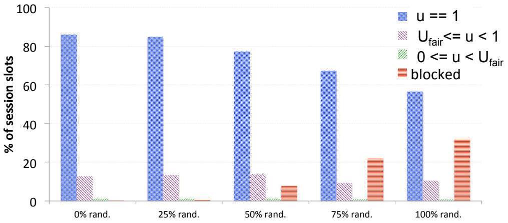


Fig. 11: Utility with different mismatch levels

performs well with no blocked requests. It is clear that by increasing $X\%$, more requests are blocked as we increase the level of mismatch between local supply and demand. It is noted that because we use a 130% $\frac{supply}{demand}$ ratio, the scenario "25% rand." is within an acceptable range of mismatch and the solution is close to the "0% rand." case.

E. Impact of inaccuracy in demand forecast

A central prerequisite for our model is the existence of a forecasting demand component that provides an input to the optimization algorithm. We discuss in this section the robustness of our solution vs. inaccuracy in forecasting demand. As shown in Fig. 11 (distributed algorithm with $N = 5\%$), the cases "50% rand." and "75% rand." respectively have around 8% and 22% of blocked user requests. On the other hand, in Fig. 7, with $N = 5\%$, there are around 18% of requests are blocked. This would mean that our results for the distributed algorithm in this paper (except Fig. 11) is corresponding to $50\% - 75\%$ inaccuracy in the forecasting demand compared to the perfect allocation case. Therefore, this mismatch leads to worse solutions. However, as shown in Fig. 7, we still can find good solution (small fraction of blocked users) for the distributed algorithm if the visibility set is large enough, for instance $N \geq 20\%$.

V. Related Work

Server selection: our work is closely related to recent work on optimizing performance-cost for server selection [8], [9]. For example, Wendell et al. [8] introduce DONAR - a decentralized replica-selection system that considers client locality, server load, and policy preferences. Like DONAR, our model can perform balancing client requests across replicas by manually setting capacity cap at each execution zone. Zhang et al. [9] focus on optimizing cost and performance in online service provider networks. The objective is to search for the optimal "sweet-spot" in the performance-cost Pareto front. Auspice [17] uses a heuristic placement algorithm to determine the locations of active replicas so as to minimize client-perceived latency. In general, these works use the classical closest method saying that the closer the servers are, the better QoS users can perceive. Our study can be a complementary for previous work as we define utility, a more general framework to qualify QoS compared to the classical closest approach. In addition, our polynomial algorithm can perform optimization for multi-services at the same time.

Network latency and traffic demand estimation: recent works have shown that the IP geolocation of the user provides accurate and predictable network latency [18]. This has been confirmed not only by third-party datasets such as Peerwise [19] and iPlane [20], but also by our own extensive active measurements [21], [22]. On the other hand, work in literature shows that client request rate can be sufficiently predictable under short interval (e.g. 10 minutes [8]). These works are useful as they provide accurate inputs for our optimization model.

VI. Conclusions and Future Work

This paper presented *utility-maximizing server selection*, a novel method to implement service instance selection that allows for trading-off user QoS with traffic cost. Compared with the classical closest approach, our utility framework allows reducing blocking probability while maintaining good utility for users. As further work, we are working on the modeling of incentives between application and network providers and how to address the scenarios where the ideal choice of server is not the same for both stakeholders. In addition, we are planning to extend the utility function to support more QoS metrics.

Acknowledgment

This research has received funding from the Seventh Framework Programme (FP7/2007-2013) of the European Union, through the FUSION (grant agreement 318205) projects.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *MCC*, 2012.

[2] I. Poese, G. Smaragdakis, B. Frank, S. Uhlig, B. Ager, and A. Feldmann, "Improving Content Delivery with PaDIS," *IEEE Internet Computing*, vol. 16, no. 3, pp. 46–52, 2011.

[3] M. Stone and B. Moore, "Tolerable Hearing Aid Delays. Est. of Limits Imposed by the Auditory Path Alone using Simulated Hearing Losses," *Ear and Hearing*, vol. 20, no. 3, 1999.

[4] M. A. Khan and U. Toseef, "User Utility Function as Quality of Experience (QoE)," in *ICN*, 2011.

[5] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility-based Placement of Dynamic Web Applications with Fairness Goals," in *NOMS*, 2008.

[6] "Recommendation p.800 (08/96)," http://www.itu.int/rec/T-REC-P.800-199608-I/en.

[7] J. Nielsen, "Usability Engineering: Response Times: The Three Important Limits," 1993.

[8] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford., "DONAR: Decentralized Server Selection for Cloud Services," in *SIGCOMM*, 2010.

[9] Z. Zhang, Y. Hu, M.Zhang, R.Mahajan, A. Greeberg, and B. Christian, "Optimizing Cost and Performance Online Service Provider Networks," in *NSDI*, 2010.

[10] H. Xu and B. Li, "Joint Request Mapping and Response Routing for Geo-distributed Cloud Services," in *INFOCOM*, 2013.

[11] S. Boyd and A. Mutapcic, "Subgradient Methods," in *Lecture notes of EE364b, Stanford University*, 2006.

[12] Www-01.ibm.com/software/commerce/optimization/cplex-optimizer.

[13] Http://www.datacentermap.com/.

[14] Https://github.com/richardclegg/multiuservideostream.

[15] G. V. Brummelen, *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton Uni. Press, 2013.

[16] S. Gangam, J. Chandrashekar, I. Cunha, and J. Kurose, "Estimating TCP Latency Approximately with Passive Measurements," in *PAM*, 2013.

[17] A. Sharma, X. Tie, D. Westbrook, H. Uppal, A. Yadav, and A. Venkataramani, "A Global Name Service for a Highly Mobile Internetwork," in *SIGCOMM*, 2014.

[18] S. Agarwal and J. Lorch, "Matchmaking for Online Games and Other Latency-sensitive P2P Systems," in *SIGCOMM*, 2009.

[19] M. Lu, J. Wu, K. Peng, P. Huang, J. Yao, and H. Chen, "Design and Evaluation of a P2P IPTV System for Heterogeneous Networks," *IEEE ToM*, vol. 9, no. 8, pp. 1568–1579, 2007.

[20] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, , and A. Venkataramani, "iPlane: An information Plane for Distributed Services," in *NSDI*, 2006.

[21] R. Landa, J. T. Araujo, R. G. Clegg, E. Mykoniati, D. Griffin, and M. Rio, "The Large Scale Geography of Internet Round Trip Times," in *IFIP Networking*, 2013.

[22] ——, "Measuring the Relationships between Internet Geography and RTT," in *ICCCN*, 2013.