Cost-Efficient, Reliable, Utility-Based Session Management in the Cloud

Benjamin Byholm
Åbo Akademi University, Dept. of Information Technologies
Turku, Finland
Email: benjamin.byholm@abo.fi

Iván Porres
Åbo Akademi University, Dept. of Information Technologies
Turku, Finland
Email: ivan.porres@abo.fi

Abstract

We present a model and system for cost-efficient and reliable management of sessions in a Cloud, based on the von Neumann-Morgenstern utility theorem. Our model enables a web application provider to maximize profit while maintaining a desired quality of service. The objective is to determine whether, when, where, and how long to store a session, given multiple storage options with various properties, e.g. cost, capacity, and reliability. Reliability is affected by three factors: how often session state is stored, how many stores are used, and how reliable those stores are. To account for these factors, we use a Markovian reliability model and treat the valid storage options for each session as a von Neumann-Morgenstern lottery. We proceed by representing the resulting problem as a knapsack problem, which can be heuristically solved for a good compromise between efficiency and effectiveness. We analyze the results from a discrete-event simulation involving multiple session management policies, including two utility-based policies: a greedy heuristic policy intended to give real-time performance and a reference policy based on solving the linear programming relaxation of the knapsack problem, giving a theoretical upper bound on achievable utility. As the focus of this work is exploratory, rather than performance-based, we do not directly measure the time required for solving the model. Instead, we give the computational complexity of the algorithms. Our results indicate that otherwise unprofitable services become profitable through utility-based session management in a cloud setting. However, if the costs are much lower than the expected revenues, all policies manage to turn a profit. Different policies performed the best under different circumstances.

Keywords

Analytical models; Distributed Systems; Markov processes; Reliability, availability, and serviceability; Simulation; Utility theory; Web-based services

1. Introduction

Session state is a form of soft state that expires after a certain time interval has passed since the last request in a given session [1]. Session state is especially important in interactive web applications that provide a rich user experience. Web applications can store session information in different storage systems, e.g. local memory, a flat file system, a distributed cache, or a database. Each session store has different characteristics, e.g. available capacity, durability, and reliability. In the context of a commercial digital service, successful handling of a session can lead to some revenue for the service provider. However, in the context of cloud computing, where the application provider pays for computing resources per use, each session store also has a different cost per transaction, reducing the expected profit. The question that we face in this article is: How can an application provider maximize the expected profit by minimizing session costs?

At any given moment, a session management system may decide to store a session in one or more storage subsystems, or it can decide to delete a session. These decisions will affect the reliability, revenue, and costs of the whole web application. They must account for hardware constraints, such as how many sessions fit in a store, e.g. local memory, or how many sessions we can write to a slow store, e.g. a database, in a particular time interval. Given any number of sessions with different expected revenues and resource requirements, e.g. size, a session management system should determine an efficient allocation of sessions to stores, meeting the reliability requirements while maximizing profit. For example, at any given moment, the system should decide which sessions to keep in volatile, but fast, random-access memory (RAM) on the application server, which sessions to keep in a reliable, but slower, remote store and which to drop entirely. We consider that there is a finite amount of RAM available, operations cost money and the expected revenues differ between sessions. A cost-efficient solution to this problem makes previously unprofitable services profitable and increases profit in others.

In this paper, we develop a general utility model to solve this problem that takes reliability into account and works in other contexts than e-commerce. We achieve this by constructing a utility function, obeying the axioms of the von Neumann-Morgenstern utility theorem [2]. The utility theorem allows us to create a risk-aware agent, capable of maximizing the expected utility of the whole system. For the sake of clarity, we only present the most basic and general version of the model, which can easily be tailored to suit implementation-specific needs. We combine this utility model with a Markovian reliability model for multiple session stores, giving the risk of data loss for a session.

Based on our model, we propose a system for utility-based session management and compare 9 different session management policies and variants through discrete-event simulations and analyze the results from 6 different scenarios. Our results show that utility-based session management increases cost-efficiency, but if the ratio between expected revenue and cost of sessions is too high, the difference will be negligible. However, with a low ratio of expected revenue to cost, such as an advertisement-funded or freemium web application, utility-based session management means the difference between the application provider operating at a net loss or profit.

2. Background and Related Work

Session state is where an application maintains a user's workflow. If this state is lost, the user will perceive an application failure, so session state has to be reliably stored with high performance. Although the most efficient way to store sessions from a performance point of view is to keep them locally in memory at the application servers [1], this alone is not reliable and does not scale well, as it requires session affinity, where users are pinned to a specific application server. Session affinity violates the principle of separation of concerns, as application servers become responsible not only for application logic, but also for storing session state [1]. Session affinity also complicates load balancing, as different classes of requests may have different resource requirements and service times, but the load can only be balanced at the session level and not at the request level.

2.1. Session State Management

Several works [1], [3], [4] have studied the problem of session state management. According to Ling et al. [1], retrieval of session state should be reliable and fast [1]. However, due to its transient nature, session state does not require full atomicity, consistency, isolation, and durability (ACID) semantics [1]. Moreover, if session state is not shared among users and is accessed in a serial fashion, there is no need for explicit synchronization or locking [1]. Ling et al. [1] presented SSM, a reliable session state manager, making use of soft state through basically available, soft state with eventual consistency (BASE) semantics. SSM modeled the reliability of identical session stores as a Poisson process causing servers to fail and restart randomly. A session would only be lost if all copies of it were lost between two write cycles. Thus, system reliability was dependent on write frequency and the number of stores used.

Fox et al. [3] proposed a layered architecture for cluster-based scalable network services based on soft state and argued that many network services can trade consistency for availability by making use of BASE semantics, which are weaker than ACID. Any data semantics that are not ACID are BASE [3].

Goldberg [4] compared two approaches for state management in distributed systems: Leases, first introduced by Gray and Cheriton [5], and soft state, first introduced by Clark [6]. According to Goldberg [4], leases reduce the number of maintenance messages necessary to keep a system consistent, a limiting factor when scaling to more hosts and services. But, leases are more suited to highly heterogeneous services that join and leave the network often [4]. Soft state offers simplified maintenance, a high degree of fault tolerance and works well with BASE semantics. BASE semantics are a perfect match for the relaxed consistency offered by soft state [4].

However, soft state, in using BASE semantics, is not free of drawbacks. For example, the level of consistency, and thereby reliability, in the system is determined by the frequency of the update messages [4]. A higher frequency of update messages will lead to increased costs and overhead. Increased overhead means greater delay. Any delay when retrieving session state will increase the service time for all requests, as processing of a request cannot proceed without the required session state. Based on these observations, we decided to base our model on soft state coupled with a reliability model, similarly to Ling et al. [1]. But, because we are interested in non-identical session stores, we require a reliability model different from that of SSM [1]. Having chosen soft state as a way of managing session state, the remainder of the problem consists of choosing where to store different sessions in a cost-efficient way.

2.2. Knapsack Problems

The knapsack problems are a family of NP-hard combinatorial optimization problems. NP-hard problems are at least as hard as the hardest problems in NP [7]. From a practical point of view, for our knapsack problem, this means that exact solutions are extremely computationally expensive, even when the problem entails as few as 100 sessions. In the knapsack problem, we are given a set of n items with associated profits p and resource requirements r, the objective is to pick some of the items x so that the aggregate profit is maximized, while not exceeding the capacity R of the knapsack. The most basic form of the knapsack problem is the 0–1 knapsack problem, which can be formalized as:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \leq R \\ & x_i \in \{0, 1\} \end{aligned} \tag{1}$$

The 0–1 knapsack can be extended to multiple resources in a variant known as the multiple-choice multi-dimension knapsack problem (MMKP) [8], shown in (2). In this problem there are n groups and group i has l_i items. The objective here is to pick exactly one item from each group to maximize profit subject to the knapsack's m resource constraints $r_{ij_k} x_{ij} \leq R_k$.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{l_i} p_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{l_i} r_{ij_k} x_{ij} \leq R_k \\ & \sum_{j=1}^{l_i} x_{ij} = 1 \\ & x_{ij} \in \{0, 1\} \\ & k = 1, \ldots, m \end{aligned} \tag{2}$$

2.3. Knapsack Problems in Session Management

Solving large knapsack problems like the MMKP optimally is computationally expensive. To obtain near-optimal solutions

to the MMKP in real-time, Khan designed an efficient heuristic called HEU [8]. The heuristic solution to the MMKP, combined with the concept of utility, solved the problem of resource management within multisession adaptive multimedia systems and Khan presented a model [8] based on these concepts.

Many later works [9], [10], [11], [12], [13] extend and improve HEU. The convex hull heuristic C-HEU [9], solves the MMKP in $\mathcal{O}(nlm + nl\log l + nl\log n)$ time, where n corresponds to the number of sessions, m is the number of dimensions and l is the number of items in a group. The iterative refinement I-C-HEU [10], promises a solution to the MMKP in $\mathcal{O}\left((x+p)\,nlm + (x+p)nl\log l + nl\log(x+p)\;n\right)$ worst case time with an estimated 90 % accuracy, where p is the batch size and x is the previous solution. In real use with a fixed batch size, this appears to amount to near constant time [10]. Ykman-Couvreur et al. [12] presented the IMEC method, a faster heuristic with complexity $\mathcal{O}(m + 2nl + nl\log(nl))$. Shojaei et al. [13] showed a parametrized compositional heuristic for the MMKP with worst case complexity $\mathcal{O}\left(n\max(l_{max}\log l_{max}, \alpha^4)\right)$, where the parameter α corresponds to the maximum number of Pareto-optimal configurations considered per session and time step. The heuristic is intended for small n, scheduling applications on an embedded system. However, it does scale to larger values of n, but the IMEC method [12] provides better optimality at equal running time [13]. Projecting the resources into one dimension leads to poor results in larger problem instances. Thus, Pareto points are computed in multi-dimensional space [13].

There are also approaches not based on HEU. For example, Bateni and Hajiaghayi [14] studied the facility location problem, which would be appropriate if routing costs between servers should be accounted for. Cohen and Katzir [15] investigated the generalized maximum coverage problem (GMCP), an extension of the budgeted maximum coverage problem (BMCP), where there is a global budget which cannot be exceeded. If individual resource constraints on the agents are not necessary, e.g. by regarding resources as unlimited, the session management problem can be modeled in this way.

2.4. Von Neumann-Morgenstern Lotteries

A von Neumann-Morgenstern lottery [2] consists of mutually exclusive outcomes that may occur with a given probability. The sum of probabilities in a lottery should be equal to one. For example,

$$L = 0.20A + 0.80B \tag{3}$$

denotes a scenario where the probability of event A is $P(A) = 0.20$, the probability of event B is $P(B) = 0.80$ and exactly one of the possible outcomes will occur. The general case of a lottery L with n outcomes A_i and probabilities p_i can be expressed as:

$$\begin{aligned} L &= \sum_{i=1}^{n} p_i A_i \\ \text{subject to } &\sum_{i=1}^{n} p_i = 1 \end{aligned} \tag{4}$$

According to the von Neumann-Morgenstern utility theorem [2], an agent faced with the problem of choosing between a set of lotteries has a utility function, provided that the four axioms of the theorem are satisfied. The four axioms of the utility theorem on lotteries L, M and N are:

- *completeness* (L or M is preferred, or they are equal) $L \preceq M \vee M \preceq L$
- *transitivity* (consistent preference across 3 operations) $(L \preceq M \wedge M \preceq N) \rightarrow L \preceq N$
- *continuity* (transitive preference is continuous) $(L \preceq M \wedge M \preceq N) \rightarrow \exists p \in [0,1] pL + (1-p)N = M$
- *independence* (independence of irrelevant alternatives) $L \prec M \rightarrow \forall N \forall p \in (0,1] pL + (1-p)N \prec pM + (1-p)N$

If an agent satisfies these axioms, it has a utility function u, assigning a real value $u(A)$ to every possible outcome A, so that for any two lotteries L and M, $Eu(L)$ is the expected value of u in L, and

$$L \prec M \leftrightarrow Eu(L) < Eu(M) \tag{5}$$

By using the utility function we can determine which lotteries to play. In this paper we will model the alternatives for handling a session as von Neumann-Morgenstern lotteries. By choosing among lotteries, we can determine an allocation of sessions to stores that maximizes the aggregate system utility.

There are, however, some limitations to von Neumann-Morgenstern utility. Von Neumann and Morgenstern [2] acknowledged that nested gambling is ignored. An example of nested gambling with lotteries L and M would be $pL + (1-p)M$, which gets treated as a lottery itself. Another limitation is that utilities cannot be compared between agents X and Y with different utility functions u_X and u_Y. Expressions like $u_X(L) + u_Y(L)$ are undefined. As we use neither nested gambling nor multiple agents, these limitations do not affect us. We may design a utility function that incorporates risk aversion or diminishing returns, which could be beneficial in a session management system focusing on reliability.

2.5. Utility in Session Management

The concept of utility provides a way of accounting for preferences over a set of goods or offers. Poggi et al. [16] used machine-learning techniques to develop a model of Web user behavior. The model provides a utility-based allocation of system resources, using the expected revenue of a session in an e-commerce application. Expected revenue is the probability that a session will end in a purchase, multiplied by the value of the goods in question. Expected revenue works as a way of prioritizing sessions: if the expected revenue is low enough, a session receives no service, e.g. when a robot tries to access a website under high load.

In addition to HEU, Khan also developed a utility model [8] for the problem of resource management within multisession adaptive multimedia systems [8]. In this problem, each session provides a quality profile, describing user preferences for different operating qualities. Operating qualities are mapped to

resource requirements by a quality-resource mapping and to a session utility through a quality-utility mapping [8]. Modifying Khan's model to better suit our problem allows us to construct a utility model for session management in a cloud setting.

Khan's utility model [8] is also a source of derivative works. Yu [17] applied multicommodity flow to Khan's utility model in replacement of the MMKP, as the resulting problem can be solved more easily. We have yet to attempt this method. Akbar et al. [18] extended Khan's utility model [8] to distributed systems and formulated the underlying problem as the multiple-choice multi-dimensional multiple knapsack problem (MMMKP) [11], an extension of MMKP to multiple knapsacks, but the context was still multimedia systems.

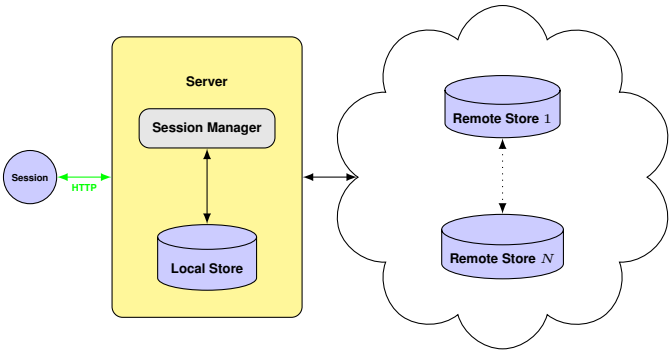


Fig. 1. A session manager runs on an application server with a local store and decides how to handle a session.

3. Session Management System

A session management system is responsible for retrieving and storing session state, as well as admission control. In a web application, the session management system lies in the critical path of requests and needs to reliably perform its tasks in near real-time without substantial overhead. In the world of Enterprise Java, session management is provided by a servlet container, which provides the run-time environment for Web components, including life cycle management, concurrency, and sessions. We have designed a profitable, reliable session management system for web applications with access to multiple session stores in a cloud setting.

The design of the session management system envisioned in this paper can be seen in Figure 1, which presents the system architecture. A session is requested through a hypertext transfer protocol (HTTP) request to a web application running on an application server. The application server has a session manager and a local memory-based cache of a finite capacity, the local store. The session manager can store session state in the local store, as well as retrieve it, unless the session has been evicted from the cache or the application server has failed and everything in the local store has been lost. Sessions may be evicted from a store either due to the session manager explicitly removing them, or implicitly through a flushing policy, e.g. least recently used (LRU), if the store has become overfull. In addition to the local store, the session manager also has access to other services providing session storage. These remote stores may have their own capacity restrictions and reliability guarantees. They usually have higher latencies.

The session manager continuously executes a time-based sampling loop. At each iteration of the loop, the sessions are redistributed according to one or more optimization criteria. If the number of sessions is large, it may not be feasible to optimize them all at once. In this case, optimization should be done incrementally in smaller batches, which will trade efficiency for effectiveness. Providing an effective and efficient way of storing sessions with minimal overhead is the goal of the proposed system. The session manager tries to achieve this goal by keeping track of sessions in the system and determining where to store them, according to a given session allocation policy. Some allocation policies may choose to drop previously admitted sessions. New sessions arrive to the session manager as HTTP requests. For new sessions, the session manager acts as a utility-based admission controller.

Utility-based admission control [16] uses the notion of utility to determine whether to admit a session or not. The system should only allow new sessions that satisfy two criteria: sufficient resources are available and the system utility will increase by admitting the session [8]. A new session should be rejected if either criterion remains unfulfilled. Khan's utility model [8] provides implicit admission control satisfying these requirements. Applying Khan's utility model to management of session state will lead to session-based admission control. Session-based admission control gives a better user experience than request-based admission control [19]. Moreover, when there are insufficient resources, taking into account the elastic nature of the cloud, it is possible to queue sessions until sufficient resources are available [19].

4. Utility Model for Session Management

Our utility model should account for the reliability of the session stores as well as for costs related to storing sessions.

There are three monetary costs directly related to maintaining session state: the read and write costs, c_r and c_w, which can consist of a transaction cost as well as a size-dependent cost, which might account for bandwidth consumed in the data transfer, and a store cost c_s dependent on the size of the stored data over time. For example, a slightly simplified version of the current prices at Amazon S3[1] gives a read cost $c_r = \$4 \times 10^{-7}$, a write cost $c_w = \$5 \times 10^{-6}$, and a store cost $c_s = \$3 \times 10^{-17}$ per byte per second. In addition to the strictly monetary costs, a session may also consume arbitrary system resources.

This paper mainly considers two stores, since Ling et al. [1] found two to three stores of greatest practical use. If the number of session stores $n = 2$, and each has one type of resource, e.g. memory, the number of resource types in the MMKP will be 2, m_a and m_b, memory from store a and b. The number of choices will be $2^n = 2^2 = 4$: store at neither, store at a,

1. http://aws.amazon.com/s3/

TABLE 1. Resource usage with two stores, a and b

$\frac{a}{b}$	0	1
0	$0m_a + 0m_b = 0$	$1m_a + 0m_b = m_a$
1	$0m_a + 1m_b = m_b$	$1m_a + 1m_b = m_a + m_b$

store at b, store at both. The resource consumption of the corresponding choices is given in Table 1.

For each session, we have a set of lotteries corresponding to the ways of handling the session. In the case of 2 storage levels, we have 4 lotteries: L_0 dropping the session, L_1 keeping the session in memory on the application server, L_2 persisting the session in the reliable store or L_3 keeping the session in memory while also persisting it in the reliable store. Each lottery L_i also has a utility function u_i. Due to assignment restrictions, all lotteries may not be playable at all times. The agent must always choose a playable lottery.

Let us consider the reliable store ideal for now, having a failure rate of 0, and the application server having a failure rate μ. The relationship between mean time to failure (MTTF) and availability can be modeled as follows: Given a MTTF of $\frac{1}{\mu_i}$, store failure can be modeled as a Poisson process $F(t)$ with rate μ_i. Similarly, writes for a user's data can be modeled as a Poisson process with rate λ. This rate affects the session expiration time, which will be $\frac{1}{\lambda}$. A session needs to exist for a time interval τ, the predefined expiration time. Thus, the write rate λ has to be at least high enough to satisfy $\frac{1}{\lambda} \leq \tau$. We will now show how to construct a utility function. To ease understanding for the reader, we only present a simple, risk-neutral variant in this section. In practice, the reliable store may not be ideal and there will be overhead costs.

If we drop the session, we will almost never profit, more formally expressed as

$$Drop: \quad u_0(s_i) = 0 \tag{6}$$

If we decide to keep the session without persisting, we will gain $v(s_i)$ revenue with probability $P(F(t) = 0)$ of the session not being lost, so

$$Local: \quad u_1(s_i) = v(s_i)P(F(t) = 0) \tag{7}$$

If we decide to persist the session and remove it from local memory, we will gain $v(s_i)$ revenue at cost $\lambda t_s(c_r + c_w)$ of reading and writing the session and cost $c_s t_s s(s_i)$ of storing the session for t_s seconds, the sampling interval. This choice amounts to

$$Remote: \quad u_2(s_i) = v(s_i) - \lambda t_s(c_r + c_w) - c_s t_s s(s_i) \tag{8}$$

Finally, when storing in both locations, we will gain $v(s_i)$ revenue at cost $\lambda t_s(c_w + P(F(t) > 0)c_r)$ of reading and writing to the session and cost $c_s t_s s(s_i)$ of storing it for the sample interval t_s, where $P(F(t) > 0)$ is the probability of the local store failing, giving

$$Both: \quad u_3(s_i) = v(s_i) - \lambda t_s(c_w + P(F(t) > 0)c_r) - c_s t_s s(s_i) \tag{9}$$

The equations for three storage levels can be derived in a similar fashion.

4.1. Optimizing System Utility

An important task for the session management system, as presented in this article, is to maximize the system utility. At any given time, the system utility is defined as the sum of the utility of all the sessions in the system.

Initially, the session management problem resembles the maximum generalized assignment problem (GAP), an APX-hard combinatorial optimization problem [20]. In the GAP, we have an arbitrary number of agents with finite resources and an arbitrary number of tasks with resource requirements. We are able to assign any task to any agent at cost and profit dependent on the assignment, provided that the aggregate resource consumption of tasks assigned to an agent does not exceed the amount of resources available. We require an assignment meeting all resource constraints while maximizing system profit. However, we can reduce the complexity of the problem by limiting the number of agents to a suitable constant. Although the number of possible assignments grows exponentially with the number of agents, there are only $2^2 = 4$ possible assignments with 2 stores and $2^3 = 8$ with 3. Choosing between assignments, we can express our problem as the NP-hard MMKP, described in Section 2.2.

Let there be n sessions, each having a set of lotteries $\mathcal{L}_i$, e.g. $\mathcal{L}_i = \{L_0, L_1, L_2, L_3\}$. Let $l_i = |\mathcal{L}_i|$ and by transforming the goal of (2), we can express the goal of the session management problem as:

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{l_i} \mathcal{L}_{ij} x_{ij} \tag{10}$$

Since each lottery has an associated utility function u, (10) can also be expressed as:

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{l_i} u_{ij} x_{ij} \tag{11}$$

All the resource constraints R_k in (2) must still be satisfied, so the final equation, where there are m resources with capacities R_k and each session i, using lottery j, consumes r_{ij_k} of resource k is:

$$\begin{aligned} \text{maximize} & \sum_{i=1}^{n} \sum_{j=1}^{l_i} u_{ij} x_{ij} \\ \text{subject to} & \sum_{i=1}^{n} \sum_{j=1}^{l_i} r_{ij_k} x_{ij} \leq R_k \\ & \sum_{j=1}^{l_i} x_{ij} = 1 \\ & x_{ij} \in \{0, 1\} \\ & k = 1, \ldots, m \end{aligned} \tag{12}$$

Because the state of the system varies frequently when sessions enter and exit the system, the model needs to be solved and re-solved in soft real-time. Frequently solving and re-solving the model means that the effort of obtaining an

optimal solution will not be worth the returns, as old solutions cannot be reused [21]. Our conjecture is that a heuristic or approximation approach will suffice and give results that are good enough in reasonable time. We present these approaches in the next section.

5. Session Allocation Policies

We present two utility-based approaches: a simple, greedy algorithm based on aggregate resource consumption, and a linear programming (LP)-relaxation of the corresponding MMKP. We will evaluate 2 variants of these algorithms: allowing sessions to be dropped and without dropping. For comparison, we have developed 3 naïve session management algorithms that always choose the same action (always local, remote, or both) and one random algorithm, of which we also made an alternative version that would not drop sessions. These algorithms and policies add up to 9 in total.

5.1. LP-Relaxed MMKP

Optimally solving large instances of the MMKP is unfeasible. For this reason, we have decided to use CPLEX[2] to solve the LP-relaxation of the MMKP instead, which can be done quickly and provides an upper bound on the achievable utility. However, the maximum utility actually achievable is likely to be somewhat lower than this bound due to integrality constraints. The solution is rounded to the nearest integer solution, so that it can be realized in the actual system, but we record the utility from the original, relaxed assignment. Similarly, we may disallow dropping sessions in a variant of the algorithm and instead choose the second most preferred alternative. In this case, we do however account for the altered preference when computing the utility. All this serves to still give us the upper bound on achievable utility at that moment, while proceeding to the next iteration of session management. While doing so does affect future state of the system, we assume this error to be negligible, taking into account that the algorithm runs on-line, only optimizing at the current problem instance, not accounting for possible future instances.

5.2. Greedy Algorithm

The classical greedy algorithm solves knapsack problems by sorting the sessions in descending order on $\frac{u_i}{r_i}$, the ratio of utility to resource consumption, as the densest items will come first this way [10]. While the greedy algorithm performs quite well in the continuous case, where fractional allocations are allowed, for a 0–1 knapsack as described in (1), the greedy algorithm does not perform so well. However, it is fast since it has the complexity of the chosen sorting algorithm. Toyoda [22] proposed a measurement called aggregate resource consumption for applying the greedy method to the multi-dimensional knapsack problem (MDKP) [10]. Khan [8] applied the concept to pick a new candidate item in a group when solving the MMKP. Toyoda [22] defines the aggregate resource consumption of choice j for session i as:

$$a_{ij} = \frac{\sum_{k=1}^{n} r_{ij_k} C_k}{|C|} \tag{13}$$

$$|C| = \sqrt{\sum_{k=1}^{n} C_k^2} \tag{14}$$

Where r_{ij_k} is the amount of resource k consumed by session i for choice j and C_k is the amount of resource k consumed by the currently selected sessions. Our greedy algorithm works by choosing the alternative with the highest density $\frac{u_j(s_i)}{a_{ij}}$ for each session s_i. The variant without dropping works similarly, but selects the second best choice if the preferred choice is dropping. The greedy algorithm is defined as:

1: **for all** s_i **do**
2: $\quad g \leftarrow sort(s_i, \frac{u_j(s_i)}{a_{ij}})$
3: $\quad select(g_0)$
4: **end for**

5.3. Naïve Algorithms

As described in Section 4, there are 4 alternatives when dealing with a session in a system with 2 stores: drop the session, write to local, write to remote, and write to both. We can construct a naïve session management algorithm based on each of these alternatives. However, the algorithm based only on dropping the session would not be very interesting, so we decided to replace that with a uniformly random algorithm instead. Thus we created the four naïve session management algorithms: random choice, always using the local store, always using the remote store, and always writing to both. As the random algorithm had the possibility of dropping sessions, we also created a variant of it that disallowed dropping sessions by making a random choice only among the three other options.

6. Evaluation Using Discrete-Event Simulations

We used SimPy[3] to develop a discrete-event simulation of a web application using the proposed system. With the simulation, we analyzed session management strategies to determine which is better with respect to system utility from the point of view of a web application provider.

6.1. Setting up the Experiment

We prepared 9 policy treatments: greedy, greedy without dropping, LP-relaxation of the MMKP, LP-relaxation of the MMKP without dropping, always writing to both, always local, always remote, random, and random without dropping. The 2 revenue treatments were exponentially distributed revenues and identical revenues. The 3 revenue/cost ratio treatments were:

2. http://www.ibm.com/software/commerce/optimization/cplex-optimizer/

3. http://simpy.sourceforge.net/

low, $10\times$ higher, and $1000\times$ higher. The base revenue for a session $v(s_i)$ was an exponentially distributed step function with mean $\mu_v = \$1 \times 10^{-4}$ and rate $\lambda_v = \frac{1}{\mu_v}$ in the interval $[\$0, \$1 \times 10^{-4}]$, starting at $v(s_i) = \$1 \times 10^{-5}$ with probability $P(C) = 0.25$ of growing or shrinking:

function ExpGrowth($x, \lambda, minimum, maximum$)
  $X \sim Exp(\lambda)$
  $x \leftarrow x \pm X$
  $x \leftarrow max(minimum, x)$
  $x \leftarrow min(maximum, x)$
  return x
end function
function StepExpRevenue(r, λ_v)
  $X \sim U(0, 1)$
  if $X < 0.25$ **then**
    $r \leftarrow ExpGrowth(r, \lambda_v, 0, \$1 \times 10^{-4})$
  end if
  return r
end function

The local store had a capacity $m_{\text{local}} = 8\,\text{MiB}$ with an LRU flushing policy implemented by random sampling of 3 sessions. While this might seem too low, it serves to show the effects experienced in a setting where the memory on the application server is not enough, possibly due to using all available memory for the actual purposes of an application server and not for storing sessions, in a reasonable amount of time required for running the simulations. The average total size of the user sessions is roughly $50\,\text{MiB}$, so it actually constitutes $20\ \%$ of the system. Session sizes were modeled in a similar fashion to expected revenues, stepwise exponentially distributed with mean $\mu_s = 256\,\text{KiB}$ and rate $\lambda_s = \frac{1}{\mu_s}$ in the interval $[1, 8192]$ KiB:

function StepExpSize(s, λ_s)
  $X \sim U(0, 1)$
  if $X < 0.25$ **then**
    $s \leftarrow ExpGrowth(s, \lambda_s, 1\,\text{KiB}, 8192\,\text{KiB})$
  end if
  return s
end function

Writing to the remote session store cost $c_w = \$5 \times 10^{-6}$, reading cost $c_r = \$4 \times 10^{-7}$, and data in the store cost $c_s = \$3.413 \times 10^{-17}\,\text{B}^{-1}\,\text{s}^{-1}$, the prices of Amazon S3. The observed variables were: system size, system revenue, and system utility. The system was simulated as an open network using exponential distributions with arrival rate $\lambda = \frac{1}{3\,\text{s}}$ and decay rate $\mu = \frac{1}{5\,\text{min}}$. The sessions had write rate $\alpha = \frac{1}{10\,\text{s}}$ and the failure rate for the local store was $\beta = \frac{1}{1\,\text{h}}$. Every experiment lasted 6 hours sampled at interval $t_s = 1\,\text{min}$.

6.2. Results

We simulated all 9 policies with 2 revenue treatments: exponentially distributed revenues and identical revenues and 3 revenue / cost ratio treatments. The results are represented in Figures 2, 3 and 4. In the legend of the figures, the ND-modifier indicates that a policy disallowed dropping sessions, while the 2-suffix indicates that the session revenues remained constant at the initial revenue $v(s_i) = \$1 \times 10^{-5}$. The results are shown as pairs of policies subject to both revenue treatments.

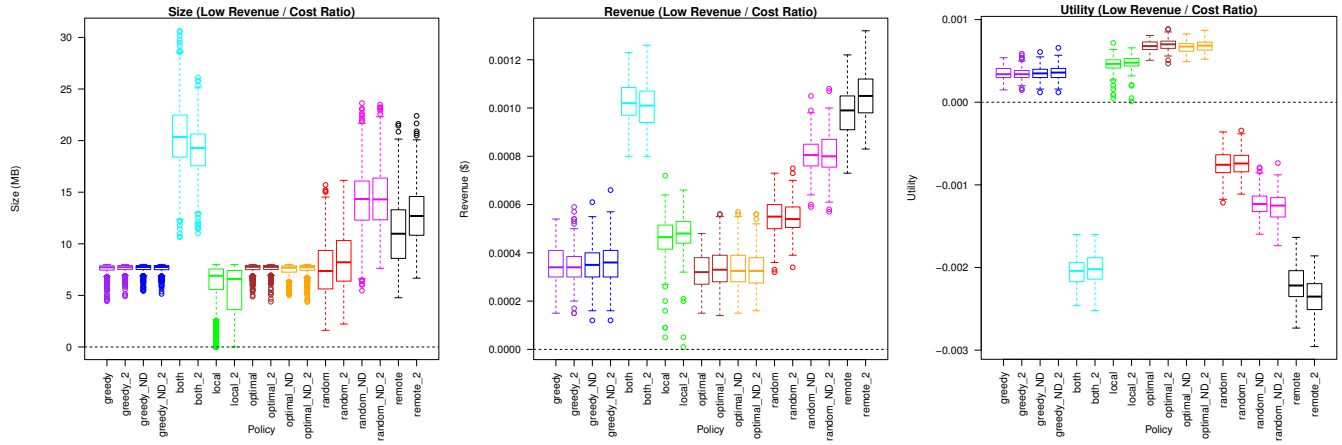
As can be seen from the system size plots in Figures 2a, 3a and 4a, the random and the greedy policies, as well as the local and the remote policy were consistent across all revenue/cost ratios, whereas the optimal policies grew in size as the revenue/cost ratio increased. The system revenues in Figures 2b, 3b and 4b show a similar pattern as the system sizes, usually with slightly more revenue when the revenue distribution was identical revenues. The real difference can be seen in Figures 2c, 3c and 4c, which show the system utilities.

With a low revenue/cost ratio as in Figure 2c, local was the only feasible approach not based on utility. The other non-utility-based approaches operated at a loss. At 10 times higher revenue/cost ratio, like Figure 3c, no approach operated at a loss, but the optimal and the policy storing to both achieved the highest utility, with the greedy policies achieving low utility, although they appear more stable than the local policy. The remote policy performed among the best, contrary to the previous scenario. Finally, with 1000 times higher revenue/cost ratio, as presented in Figure 3c, the policies performed in a similar way, but with greater disparity between the local and the both policy. The local policy now performed worst, except in the simulations with constant values, where there now was a considerable difference between results of several policies.

6.3. Analysis

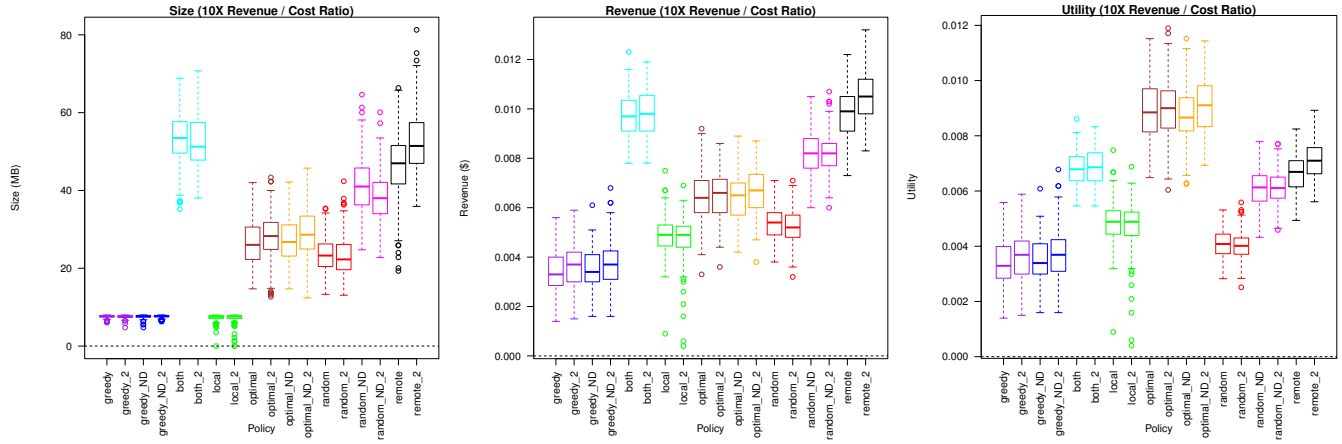
The results indicate that utility-based session management is a viable option in the general case, usually outperforming the naïve methods, but it becomes especially useful in a situation with a low revenue/cost ratio. Utility-based session management can open up new markets and mean the difference between a business venture operating at a loss or a profit. A real-world example of such a scenario would be delivering a service financed through advertisement revenue. The local policy also turned a profit under these circumstances, but is limited by the lack of space and lack of reliability inherent to relying solely on local memory and will perform worse with longer sessions. The results show many outliers for the local policy. Due to its unstable nature, it is nonviable as a real-world option.

We observe no greater difference between scenarios with constant revenue and those with stepwise exponentially distributed growth until the revenue/cost ratio reaches 1000. The only policy unaffected by the change in revenue model was the local policy, suggesting that its achievable income was limited by an external factor. As expected, a look at the size plot shows this limiting factor as the available size in the local store. The greedy policy also appears constrained in size for the same reason. It appears to have made too risky decisions. An improved algorithm should perform better, as indicated by the optimal reference policies. Both models should



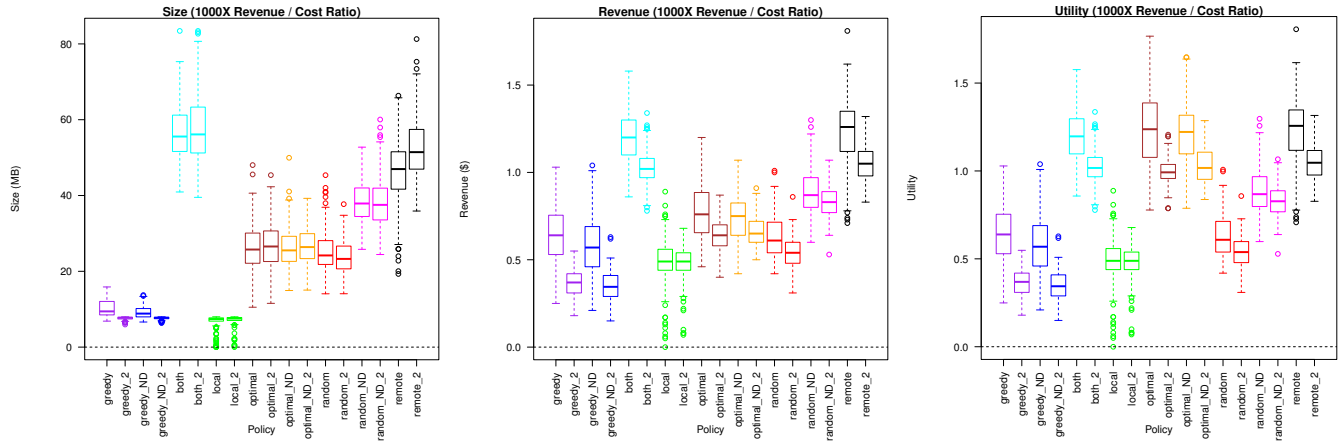
(a) (b) (c)

Fig. 2. System size, revenue, and utility with low revenue/cost ratio. ND indicates that a policy disallowed dropping sessions; the 2-suffix indicates that the session revenues remained constant at the initial revenue $v(s_i)$.



(a) (b) (c)

Fig. 3. System size, revenue, and utility with $10\times$ revenue/cost ratio. ND indicates that a policy disallowed dropping sessions; the 2-suffix indicates that the session revenues remained constant at the initial revenue $v(s_i)$.



(a) (b) (c)

Fig. 4. System size, revenue, and utility with $1000\times$ revenue/cost ratio. ND indicates that a policy disallowed dropping sessions; the 2-suffix indicates that the session revenues remained constant at the initial revenue $v(s_i)$.

work, dependent on which best suits the application provider's needs. The consistency in the results shows that the model works regardless of whether expected revenues grow or remain constant. But, as the revenue/cost ratio increases, the choice of strategy becomes less important. When comparing system sizes with their respective utilities, it shows that splurging on size becomes less of an issue as the cost shrinks in comparison to the revenue. The same applies when comparing system sizes with system revenues.

7. Increased Reliability with Multiple Stores

So far we have assumed that our system has only two stores and that the second level store is ideal. However, this assumption may be too restrictive for practical use. It is possible that an application provider wants a collection of cheaper but unreliable stores. In this section, we remove that assumption and discuss how to increase reliability by considering more stores and how this affects the utility model.

The relationship between mean time to failure (MTTF) and availability can be modeled as follows: Given a MTTF of $\frac{1}{\mu_i}$, store failure can be modeled as a Poisson process with rate μ_i. Similarly, writes for a user's data can be modeled as a Poisson process with rate λ. The session expiration time will then be $\frac{1}{\lambda}$. A session needs to exist for a time interval τ, the predefined expiration time. Thus, the write rate λ has to be at least high enough to satisfy $\frac{1}{\lambda} \leq \tau$. The ratio of MTTF to mean time between writes is given by $\rho_i = \frac{\lambda}{\mu_i}$, the operability ratio [23].

A session will be lost only if all copies of the session are lost. All copies of a session are recreated on each write. In other words: given n copies of a session, it will not be lost if at most $n-1$ copies are lost before the next write. According to [23], we can define unavailability of a unit as

$$U_i = 1 - A_i = \frac{1}{\rho_i + 1} \tag{15}$$

and unavailability of a session consisting of n copies as

$$U_s = \prod_{i=1}^{n} U_i \tag{16}$$

to obtain availability of that session as

$$A_s = 1 - \prod_{i=1}^{n} U_i \tag{17}$$

By expanding the polynomial

$$\prod_{i=1}^{n} (A_i + U_i) = 1 \tag{18}$$

we can rewrite (17), providing the probabilities of being in any one of the possible states. By adding the probabilities of the acceptable states, we obtain the availability of a session. We can now compute the operability ratio required to achieve a desired reliability guarantee with a given amount of copies at unequal stores. Figure 5 shows the probability of data loss and Table 2 shows the necessary ratio for one to three copies

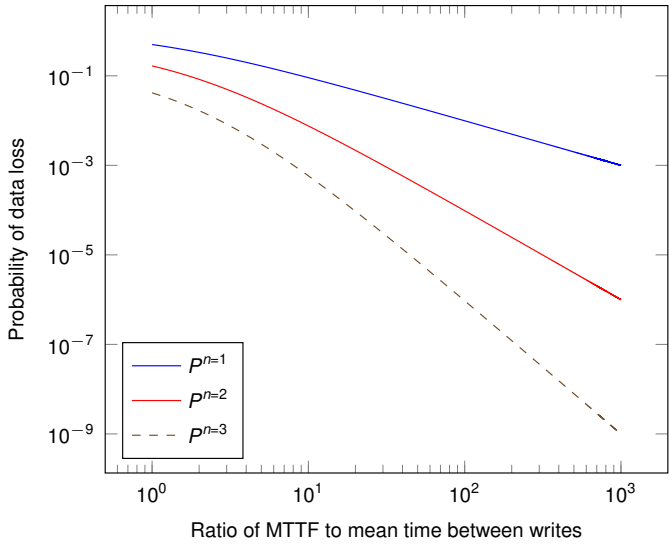


Fig. 5. Probability of data loss for a given ratio of MTTF to mean time between writes, given n identical copies.

TABLE 2. Required ratio ρ of failure rate μ to write rate λ for different data loss probabilities P with n uniform copies

P_{loss}	$\rho_{n=2}$	$\rho_{n=3}$
10^{-1}	3	2
10^{-2}	9	4
10^{-3}	31	9
10^{-4}	99	21
10^{-5}	316	46
10^{-6}	999	99

with different reliability guarantees at equal session stores. Two to three copies is most likely in practical use [1].

As mentioned in Section 1, the session management policy affects both reliability and cost. We have shown that it is rather easy to increase reliability by using more stores, but doing so will lead to increased costs. In the general case, with n stores for every session s_i, there are 2^n lotteries $L_0 \ldots L_{2^n-1}$. Every lottery L_{ij} has an expected utility u_{ij} given by

$$u_{ij} = v(s_i) \times (1 - P_{\text{fail}}) - \sum c_w - \sum c_r - \sum c_s \tag{19}$$

Where $\sum c_w$ is the cost of writing to all the stores used, $\sum c_r$ is the cost of reading, which might increase if a cheaper store loses data that is available at another location, and $\sum c_s$ is the cost of storing data over time at the stores in question. Because the read cost may vary, it can be modeled in greater detail by accounting for individual data loss probabilities of the affected stores, like we did in the example with two stores. The binomial formula $\binom{n}{k}$ tells us, for n stores, how many lotteries use a given number k of stores. Likewise, we can use the binary representation of a lottery ordinal to represent which stores each lottery uses. For example, with 3 stores, lottery 6 would use the second and third store, as its binary representation is 110_2. Accordingly, lottery 0 stores nowhere.

Assuming that we have a preference for reading from stores, so that the first store is preferred to all others, and the second

store is preferred to the third, the total cost of reading is

$$\sum c_r = p_1 c_{r1} + (1-p_1)(p_2 c_{r2} + (1-p_2)(p_3 c_{r3} + \ldots)) \quad (20)$$

Where p_n is the probability of store n surviving. Storage cost depends on how many stores survive. A store can be in two states, failed with probability $(1 - p_n)$ or functional with probability p_n. Taking all combinations of states gives 2^n possible outcomes, as usual we ignore the null option, adding them up gives the total cost. For 2 stores the store cost is

$$\sum c_s = p_1(1-p_2)c_{s1} + (1-p_1)p_2 c_{s2} + p_1 p_2 (c_{s1} + c_{s2}) \quad (21)$$

8. Conclusions

We presented a model based on von Neumann-Morgenstern utility [2] for management of session state. The concept of utility gave us a way of taking risk into account when making decisions, by using a probabilistic reliability model for session stores in a cloud setting. In this way, we were able to formulate the session management problem as an MMKP, which can be heuristically solved in real-time. We designed a utility-based session management system based on our model and presented its proposed architecture along with two utility-based session management policies, a greedy policy and a reference policy based on solving the LP-relaxation of the MMKP. By constructing a discrete-event simulation and comparing 9 different policies and variants in 6 different scenarios, we have found a significant difference between utility-based and naïve session management strategies.

Our results show that otherwise unprofitable services become profitable through utility-based session management. In a scenario with a low revenue/cost ratio, such as an advertisement-funded website, expected profits varied from $-\$0.002$ to $\$0.001$ between different session management policies. When the costs were three orders of magnitude lower than the revenues of sessions, all policies managed to turn a profit. Different policies performed the best under different circumstances. When relative costs are low, always storing at both works well, as the extra cost is insignificant. As our work was of an exploratory nature, we did not directly measure the execution time of the algorithms. Instead, we provided the computational complexity of the algorithms. We did not consider possible penalties for losing a session, but this could easily be added in an implementation.

Future work includes implementing the proposed session manager in a real cloud and performing an evaluation through realistic experiments. We should also study various utility functions with different risk attitudes. We seek to determine the feasibility of solving the model in real-time using different approaches for a realistic problem size to ensure a reliable and scalable solution with satisfactory efficiency and effectiveness.

Acknowledgments

This work was supported by the Cloud Software Finland research project. We also wish to thank Adnan Ashraf at Åbo Akademi University for helpful feedback on this paper.

References

[1] B. C. Ling, E. Kiciman, and A. Fox, "Session state: Beyond soft state," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 295–308.

[2] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 3rd ed. Princeton, NJ: Princeton University Press, 1953.

[3] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," *SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, pp. 78–91, Oct. 1997.

[4] D. W. Goldberg, "State considerations in distributed systems," *Crossroads*, vol. 15, no. 3, pp. 7–11, Mar. 2009.

[5] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," *SIGOPS Oper. Syst. Rev.*, vol. 23, no. 5, pp. 202–210, Nov. 1989.

[6] D. Clark, "The design philosophy of the DARPA internet protocols," *SIGCOMM Comput. Comm. Rev.*, vol. 18, no. 4, pp. 106–114, 1988.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[8] M. S. Khan, "Quality adaptation in a multisession multimedia system: Model, algorithms, and architecture," Ph.D. dissertation, University of Victoria, Victoria, B.C., Canada, Canada, 1998, aAINQ36645.

[9] M. M. Akbar, M. S. Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja, "Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls," *Comput. Oper. Res.*, vol. 33, no. 5, pp. 1259–1273, May 2006.

[10] M. Rouf, "Incremental convex hull approach applied to optimal admission control & QoS adaptation in multimedia systems," Bachelor's Thesis, Bangladesh University of Engineering and Technology, 2005.

[11] M. M. Akbar, E. G. Manning, G. C. Shoja, S. Shelford, and T. Hossain, "A distributed heuristic solution using arbitration for the MMMKP," in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*, ser. AusPDC '10. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2010, pp. 31–40.

[12] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 3, pp. 35:1–35:16, May 2011.

[13] H. Shojaei, A. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes, "A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, 2009, pp. 917–922.

[14] M. Bateni and M. Hajiaghayi, "Assignment problem in content distribution networks: Unsplittable hard-capacitated facility location," *ACM T. Algorithms*, vol. 8, no. 3, pp. 20:1–20:19, Jul. 2012.

[15] R. Cohen and L. Katzir, "The generalized maximum coverage problem," *Information Processing Letters*, vol. 108, no. 1, pp. 15–22, 2008.

[16] N. Poggi, T. Moreno, J. Berral, R. Gavaldà, and J. Torres, "Self-adaptive utility-based web session management," *Comput. Netw.*, vol. 53, no. 10, pp. 1712–1721, Jul. 2009.

[17] L. L. Yu, "Multicommodity flow applied to the utility model: A heuristic approach to service level agreements in packet networks," Master's thesis, University of Victoria, 2003.

[18] M. M. Akbar, E. G. Manning, and G. C. Shoja, "Distributed utility model for distributed multimedia server system," in *Proceedings of the Fifth International Conference on Computer and Information Technology*, vol. 2. Citeseer, 2002, pp. 108–112.

[19] A. Ashraf, B. Byholm, and I. Porres, "A session-based adaptive admission control approach for virtualized application servers," in *The 5th IEEE/ACM International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2012, pp. 65–72.

[20] D. W. Pentico, "Assignment problems: A golden anniversary survey," *Eur. J. Oper. Res.*, vol. 176, no. 2, pp. 774–793, 2007.

[21] C. Archetti, L. Bertazzi, and M. G. Speranza, "Reoptimizing the 0-1 knapsack problem," *Discrete Appl. Math.*, vol. 158, no. 17, pp. 1879–1887, Oct. 2010.

[22] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems," *Management Science*, vol. 21, no. 12, pp. 1417–1427, 1975.

[23] *MIL-HDBK-338B*, Electronic Reliability Design Handbook, Air Force Research Laboratory, 525 Brooks Road, Rome, NY 13441-4505, 1998.