Proceedings of the 7th International Workshop on Economics Driven Software Engineering Research (EDSER-7),
affiliated with the 27th International Conference on Software Engineering (ICSE), St. Louis, MO, May, 2005.

Anticipatory Configuration of Resource-aware Applications

Vahe Poladian, João Sousa, Frank Padberg, Mary Shaw
School of Computer Science, Carnegie Mellon University
{vahe.poladian, jpsousa, mary.shaw} at cs.cmu.edu, padberg at ira.uka.de

ABSTRACT

We propose an improved approach to dynamic configuration of resource-aware applications. The new *anticipatory model of configuration* maximizes utility based on three inputs: user preferences, application capability profiles, and resource availability. In this respect, the proposed model is similar to a model of configuration described in [2]. However, the latter addresses the dynamic nature of the problem by *reacting* to changes (such as decrease in resource availability), and maximizes the utility in a point-wise manner. The newly proposed anticipatory approach explicitly models the duration of the task and leverages possible information about the future (such as stochastic resource availability over the expected duration of the task).

We expect that the anticipatory model will improve user's utility, conserve scarce resources, and reduce the amount of disruption to the user resulting from changes when compared to the reactive model. However, the optimization problem underlying the anticipatory model is computationally more difficult than the problem underlying the reactive model. We would like to investigate if the anticipatory approach is feasible and efficient in practice while delivering the above-mentioned improvements. In this paper, we carefully state the model of anticipatory configuration, highlight the sources of complexity in the problem, propose an algorithm to the anticipatory configuration problem, and provide a roadmap for research.

Keywords

Multi-dimensional utility, resource, stochastic process, dynamic configuration, engineering for value.

1. INTRODUCTION

In [2], we presented a model of dynamic configuration. This model takes three inputs and computes optimal configurations. First input is a task request, which is an abstract description of the capabilities that a user needs (e.g., "browse web" and "play video") and preferences for the associated quality dimensions (e.g., "latency", "frame rate", "color"). Second is a collection of capability profiles of adaptive, resource-aware applications that describe the runtime resource requirements for the various level of quality of service that these applications can provide. And third is a snapshot of available supply of scarce resources, such as bandwidth and CPU. The *reactive* model maximizes the function of user's preferences (in other words, user's utility), subject to application capabilities and available resources. The resulting algorithm computes a near-optimal configuration: (1) an assignment of concrete applications to abstract needs in the task, and (2) quality level set-points for each application. For each selected application, the configuration also contains resource allocation limit that can be used by the application and supporting mechanisms for choosing adaptation strategies.

Recognizing that the computing environment changes over time (e.g., resource availability might change, applications might fail), the model in [2] proposes *reacting to changes as they occur*, effectively maximizing utility in a point-wise manner. To mitigate against the possibility of frequently switching applications the reactive model applies penalties every time a running application is switched without user's explicit request.

The reactive model of configuration has one primary drawback. Since it computes near-optimal configurations point-wise, the resulting sequence of near-optimal solutions might fall far from being globally optimal. In other words, the utility that the user gets over the duration of the task might be far less than globally optimal, even though each configuration is near-optimal at the time it is selected.

To alleviate this problem, we propose an anticipatory model, which has the following new elements:

- Explicitly considers the *duration* of the task,
- Models *utility accrual* over time,

- Models the availability of resources as stochastic processes,
- Considers non-perishable resources such as battery.

The anticipatory model of configuration maximizes the *expected accrued utility* over the duration of the task given the following inputs: (1) expected duration of the task, (2) user's preferences, (3) application profiles, and (4) stochastic processes describing resource availability over time. Furthermore, the anticipatory model can treat the duration of the task as a decision variable, and optimize a measure of utility that combines both task duration and dimensions of output quality in the presence of non-perishable resource constraints (for example, battery).

We propose to model time discretely and use a dynamic programming algorithm to solve the optimization problem. While our solution is simple, the large number of states that the algorithm needs to consider might make it infeasible.

We expect that an infrastructure based on the anticipatory configuration model and solution can deliver benefits in the following areas of supporting a user's task:

- Improve quality of service of the task,
- Conserve scarce resources,
- Reduce configuration time and effort to start the task,
- Reduce configuration effort during the task,
- Reduce disruptive changes during the task,

We also expect that the anticipatory configuration model to perform better in one or more of the above dimensions when compared to the reactive model.

In Section 2, we present the details of the anticipatory model of configuration. In Section 3 we outline a solution to the optimization problem and present a roadmap for research.

2. THE SKETCH OF THE MODEL

2.1 Model of a User Task

A task is *T* is a set of services: $T \triangleq \{S_i, i = 1,...n\}$. Each service is an abstract description of capabilities: *type* (for example, "play video"), and quality dimensions (for example, "frame rate", "image size"). Formally, $S_i = (\tau_i, Q_i)$, where τ_i is the type of service, and Q_i is the Cartesian product of the quality dimensions, q_{ij}. Figure 1 shows a graphical representation of a task.

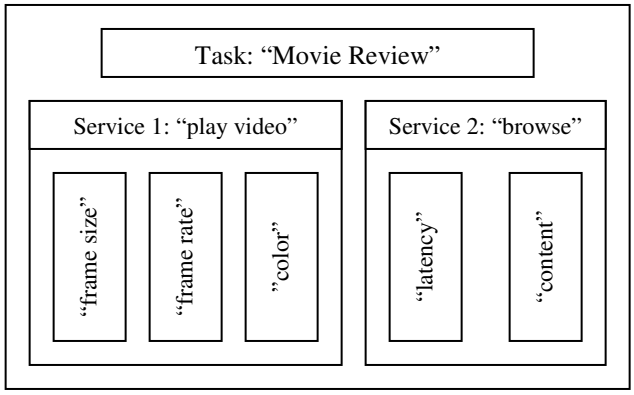

Figure 1: Visual representation of a task. A task is a set of services. Each service is described by its type and quality dimensions.

2.2 Time and Task Duration

We model time as small, discrete intervals. We assume that the state of the world *may only change* at the beginning of each interval and remains constant throughout the interval. We denote time intervals using integers. For this paper, we will use small letter *m* to index time and let the task duration go from *0* to *M*.

2.3 Utility

Utility of service S_i is computed by combining preferences for quality of service and specific applications using weigted sums. Specifically, the components of service utility are the following:

- One preference function for each quality of dimension. This can be an arbitrary function that maps a quality level to a real number,
- One preference function for the choice of specific application that provides the service type. This is typically an enumerated function,
- One penalty function that captures user's tolerance to changing the application,

These functions are combined using weighted addition.

Utility of the task is the sum of service utilities. We denote the utility of the task as U_{task}. The utility we have just described is the point-wise (instantaneous) task utility.

Let the task start at time *0* and last until time *M*. We model the utility accrual over time by taking time-discounted sum of the instantaneous utility at each time window:

$$AccU_{Task}(M) = \sum_{i=0}^{M-1} \delta^m U_{Task}(i)$$

where the instantaneous utilities on the right-hand side of the equation now have an additional argument showing time and δ is a discount factor between 0 and 1.

2.4 Application Profiles

An application profile describes the capabilities of a specific application running on a particular piece of hardware. This includes the name of the application, version, the service type it supports. The profile also includes a map that describes how efficiently the application converts resources into quality of service. Formally, this map a relation between the quality space of the service and resources demanded. Each tuple in the relation shows a possible level of service the application can deliver, if the requisite vector of resources is available for that application. Figure 2 shows the sketch of an application profile.

Application Profile Name: "QuickTime" Service Type: "play video"	QoS 1: "frame size" QoS 2: "frame rate" QoS 3: "color"
Quality to Resource Map	
100x200, 12 fps, BW	80 Kpbs, 10% CPU
100x200, 12 fps, 8bit	120 Kpbs, 12% CPU
100x200, 24 fps, BW	144 Kpbs, 14% CPU
…	…

Figure 2: A sketch of an application profile.

2.5 Resources

We model the availability of a resource as a stochastic process over time. Formally, the availability of resource *i* is a function of time R_i of a random variable. At time *m* the value $R_i|m$ is no longer random: all the uncertainty in the value of the resource has been resolved.

Because of the discrete nature of the problem, we consider discrete random processes.

2.6 Optimization Problems

Recall that a configuration is an assignment of applications to the services in the task, together with a level of quality service that each application should provide.

The optimization problem is to find a sequence of configurations that maximizes the accrued utility over the duration of the task, given the application profiles and resource availability.

Solving the problem of optimal anticipatory configuration requires solving a couple of sub-problems. Not surprisingly, these are problem instances that are solved as part of reactive optimal configuration. Next, we describe those problem instances.

Best QoS Instantaneous. Given a fixed assignment of applications, find a level of quality for each application, together with the resource allocation, such that the instantaneous task utility is maximized.

A scalable solution to this problem based on greedy approximation is given in [1]. The solution has runtime complexity O(N*LogN), where N is the aggregate number of points in the quality-resource profiles of all the applications in the assignment.

Optimal Reactive Configuration. Given multiple candidate applications for each service in a task, find a configuration such that the instantaneous task utility is maximized. An efficient solution to this problem in the context of an infrastructure for UbiComp is given in [2]. This solution relies on the solution to Best QoS Instantaneous problem instance and uses a heuristic to explore application assignments that have high potential utility.

Optimal Anticipatory Configuration. Given multiple candidate applications, a task duration, and stochastic resource availability information, find a sequence of configurations such that the accrued utility over the duration of the task is maximized.

To our knowledge, this is a new problem instance.

The optimal anticipatory configuration problem may not be computationally feasible without imposing additional restrictions. In the next section we discuss the sources of complexity in the problem and possible solutions.

3. PROPOSED SOLUTION

3.1 A Proposed Solution

We propose solving the Optimal Anticipatory Configuration problem using dynamic programming. To help illustrate the solution, let's assume for simplicity that the processes describing the availability of resources are non-random (one can view this as a degenerate case of a random process, when all the probability mass is assigned to one point).

Let the number of distinct application assignments be *P*, and the number of time windows be *M*. Let indices *p, q* range over the application assignments and index *m* range over the time windows. Let *BestQoS(p,m)* denote the solution to the Best QoS Instantaneous problem at time *m* if the application assignment chosen is *p*. In the first phase of the solution, we compute *BestQoS* solutions for all time periods *M* and all assignments *P*. The runtime complexity of computing this is O(M) * O(P) * O(BestQoS).

Next, we compute the penalty resulting from switching application combination *p* to application combination *q*. Note that such computation is independent of the resource level or the time period. This computation requires O(P*P) time, assuming the penalty computation for one pair can be in done in O(1) time.

Next, we compute optimal sequence of configurations up to time *m*, where *m* varies from *0* to *M*. Let *Seq(p,m)* denote the sequence of configurations that maximizes the accrued utility $AccU_{Task}(m)$ through time period *m*, with the restriction that at time period *m*, application assignment with index *p* is chosen (in other words, in *Seq(p,m)*, application assignment *p* is always chosen at time *m*).

Denote by *M(p,m)* the utility accrued from *Seq(p,m)*. Observe that the following recursive relationship holds:

$$M(p,m) = \max_{1<=q<=P}(M(q, m-1) + \underline{U}_{Task}(m)),$$

where $\underline{U}_{Task}(m)$ is the maximum instantaneous utility that the p^{th} application assignment can deliver, given that in the previous time window the q^{th} application assignment was chosen (in other words, $\underline{U}_{Task}(m)$ depends on the choice of *q*).

The formula above states that in order to compute maximum utility accrued from *Seq(p,m)*, we need to consider the maximum utility possible for all sequences up to time *m-1*, and then add the utility from choosing combination *p* at time *m*. That latter term, $U_{task}(m)$, depends on the combination chosen at time *m-1*.

Once we have computed the values of *M(p,m)* for all *p* and all *m*, we identify that *p'* which delivers the maximum M(p,M). We claim that the corresponding sequence *Seq(p',M)* is the solution to the Optimal Anticipatory Configuration problem.

The recursive formula lends itself to a dynamic programming solution that is linear in the number of time periods considered. At each iteration of the dynamic program, O(P) instances of Best QoS Instantaneous problem need to be solved, one for each supplier assignment. Change penalties need to be solved for each pair of assignments. However, this can be done once, because change penalties do not depend on the level of resources, as mentioned above. Thus the overall complexity of the solution is O(M) * O(P) * O(BestQoS) + O(P*P).

We made a simplifying assumption that the resource paths are non-random for the sole purpose of illustrating a dynamic programming solution. When resources are allowed to be adapted stochastic processes, the solution needs to be refined appropriately. We believe this is possible, but there are various computational issues.

3.2 Research Plan

When processes describing available resources are allowed to be stochastic, the computational complexity of the problem grows. However, it is realistic to assume that the possible resource paths and their probabilities depend only on the value of the resource in the present. Indeed, the value of bandwidth in the next instance may depend on the value of bandwidth now, but it should not depend on how we arrived at the present value. This property of stochastic processes is called Markov. This realistic assumption allows us to maintain the problem computationally feasible for interesting problem sizes.

Even with this assumption, there are a number of parameters that significantly affect the complexity of the computational problem:

- Granularity of the time window. At what granularity the problem becomes computationally infeasible?
- The branching factor of the resource paths. In other words, how many different values can the random variable describing resource availability take?
- Non-perishable resources such as battery and the relationship to other resources, such as maximum CPU available. How does this affect the computational complexity of the problem?

And lastly, we would like to compare the anticipatory and reactive models of configuration. How does the anticipatory model compare to the reactive model? Under what circumstances does the reactive model provide good-enough solutions?

We plan to investigate the boundary between computational feasibility and efficiency of the optimization problems underlying the anticipatory configuration problem. Specifically, we propose to parameterize the variables mentioned in the above list and perform experiments to determine where the problem becomes infeasible in practice. We would like to also investigate the circumstances under which the anticipatory configuration delivers really poor solution.

4. ACKNOWLEDGMENTS

This work has been funded in part by the National Science Foundation under Grant CCF-0438929, by the Sloan Software Industry Center at Carnegie Mellon, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

5. REFERENCES

[1] Chen Lee, et al. A Scalable Solution to the Multi-Resource QoS Problem. *Proc IEEE Real-Time Systems Symposium (RTSS)*, 1999.

[2] Vahe Poladian, Joãa Pedro Sousa, David Garlan, Mary Shaw. Dynamic Configuration of Resource-Aware Services. *In Proc. 26th Intl Conf. On Software Engineering (ICSE 2004)*. Edinburgh, May 2004.