IPP Hurray!

www.hurray.isep.ipp.pt

Technical Report

Dynamic Adaptation of Stability Periods for Service Level Agreements

Luis Nogueira

Luis Miguel Pinho

TR-060801

Version: 1.0

Date: August 2006

Dynamic Adaptation of Stability Periods for Service Level Agreements

Luis NOGUEIRA, Luis Miguel PINHO

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {luis, lpinho}@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

A QoS adaptation to dynamically changing system conditions that takes into consideration the user's constraints on the stability of service provisioning is presented. The goal is to allow the system to make QoS adaptation decisions in response to fluctuations in task traffic flow, under the control of the user. We pay special attention to the case where monitoring the stability period and resource load variation of Service Level Agreements for different types of services is used to dynamically adapt future stability periods, according to a feedback control scheme. System's adaptation behaviour can be configured according to a desired confidence level on future resource usage. The viability of the proposed approach is validated by preliminary experiments.

Dynamic Adaptation of Stability Periods for Service Level Agreements

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group
Polythecnic Institute of Porto, Portugal
{luis,lpinho}@dei.isep.ipp.pt

Abstract

A QoS adaptation to dynamically changing system conditions that takes into consideration the user's constraints on the stability of service provisioning is presented. The goal is to allow the system to make QoS adaptation decisions in response to fluctuations in task traffic flow, under the control of the user. We pay special attention to the case where monitoring the stability period and resource load variation of Service Level Agreements for different types of services is used to dynamically adapt future stability periods, according to a feedback control scheme. System's adaptation behaviour can be configured according to a desired confidence level on future resource usage. The viability of the proposed approach is validated by preliminary experiments.

1 Introduction

Most real-time applications have a certain degree of flexibility in terms of resource requirements. Video applications, for example, can adapt to bandwidth limitations with image compression and frame rate reduction. This adaptation process can be seen as the allocation and dynamic re-allocation of a finite amount of resources during applications' execution.

Furthermore, users can differ enormously in their service requirements as well as applications in the resources which need to be available. In [3] the authors point out that the user's QoS requirements may even change throughout a session and propose that the user should be given the opportunity to make informed decisions about application's adaptation. Therefore, there is an increasing need for customisable services that can be tailored to each user's specific requirements [9].

Our work focuses on optimising a dynamic set of tasks that can be executed at varying levels of QoS to achieve efficient resource usage that constantly adapts to devices' specific constraints, nature of executing tasks (hard real-time, soft real-time, non real-time) and dynamically changing system conditions.

We base our approach on a general form of QoS contract between service providers and users, achieved by negotiation and dictated by users' preferences [6]. Service providers allocate resources to each new set of tasks, achieving the best possible instantaneous QoS, establishing an initial Service Level Agreement (SLA) for the new task. A SLA contains a service description whose parameters are within the range of the user's desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. Once a SLA is admitted, it may be downgraded to a lower QoS level (until $L_{minimum}$ is reached) in order to accommodate new service requests or upgraded (until $L_{desired}$ is reached) on an underutilisation of the service provider.

However, frequent QoS reconfigurations may be undesirable for some users or applications. For example, in some video applications a constant frame rate may be better than a frequent variation whose average is higher than the initial contracted level of service. It is important to choose resource (re)allocation policies that beside trying to maximise the provided QoS also maximise user's influence on the variation of an initial SLA, increasing user-defined stability in service provisioning.

This paper explores these ideas and proposes a QoS adaptation mechanism that extends users' influence also to applications' adaptation during execution. Service providers propose future stability periods in response to dynamic fluctuations in task traffic flow. Proposed stability periods are compared with users' constraints for a service upgrade. As these constraints are harder to achieve it is increasingly difficult to change and stay in a better quality level.

2 Expressing desired quality and stability

There may be in the system several instances of an application used by many different users. Each of the users will have their own QoS preferences, described through a scheme that defines dimensions, attributes and values, as well as relations that maps dimensions to attributes and attributes to values [5]. A service request expresses the spectrum of acceptable QoS levels, ranging from a desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. The relative decreasing order of importance imposed in dimensions, attributes and values expresses user's preferences in a qualitative way (elements identified by lower indexes are more important than elements identified by higher indexes), eliminating the need to specify an utility value for every quality choice.

Along with his QoS preferences, the user may wish to specify a *QoS stability* personal choice influencing not only an initial QoS provisioning but also the QoS adaptation during application's execution. Possible attributes for this stability dimension are a minimum stability period Δ_{min} and a minimum increment in service's achieved utility U_{min} when upgrading to a better QoS level. As such, when upgrading to a better QoS level, system's adaptation behaviour is based around a set of user defined parameters that govern the rate of state changes. These measures can be interpreted as "do not change to a better quality state unless this gives me at least an increment in utility of U_{min} over a Δ_{min} period".

The degree of acceptability of proposed level of service and stability period is determined by measuring the distance between user's preferred values and the values proposed by a service provider [6].

3 Supporting QoS adaptation and stability

Any service provider's resource allocation policy is subject to environmental uncertainties, and for that reason, the promised SLA can never be more than an expectation of an average service quality [1]. Making an effective use of system's resources in a dynamic system is a complex and difficult task [2].

The traditional approach was that the adaptation behaviour should be integrated within applications. Although minimum system changes were required to implement QoS delivery, there are some disadvantages in that approach. Different applications running on the same system may have a different adaptive behaviour when a fluctuation of task traffic flow occurs. Some of them may require a considerable amount of resources to perform their desired adaptation, while others may not be able to perform any adaptations at all. Furthermore, the adaptation component integrated into an application is not generic and reusable.

It is therefore desirable to propose a generic mechanism that adapts application's execution to the dynamically changing system's conditions. Given the spectrum of user's acceptable QoS levels a service provider formulates the best instantaneous SLA it can offer. The local QoS optimisation (re)computes the set of QoS levels for all local tasks, including the new requested one, finding a feasible set of service configurations that maximise users' satisfaction [6].

During services' execution the task set on each node is not fixed, since tasks are continuously arriving and departing the system. The promise of service includes a stability period Δ_t, indicating that during a specific interval of time the promised level of service will be assured, either by the service proposal formulation algorithm [6] that may imply service degradation of existing tasks in order to accommodate a new service and by the QoS reconfiguration algorithm proposed in Section 4 that tries to upgrade the current SLA of previously degraded tasks.

Service stability could be achieved by using a fixed, large enough value for Δ_t, but this would then result in lack of responsiveness in adaptability. Furthermore, fixed values only make sense when there is some knowledge on the task traffic model. Promised stability periods should be determined taking into consideration the observed variations in task traffic flow and correspondent resource usage, adapting the system to environmental changes.

Although application specific mechanisms exist to allow some automatic adaptation (for example, elastic buffering in audio tools), we would like to offer a more general model to allow service providers to dynamically adjust their service provisioning to task traffic flow fluctuations. However, QoS management will not only be subject to system's specific behaviour but also to users' preferences. Our goal is to allow each service provider to make adaptation decisions in response to fluctuations in task traffic flow, but the adaptation process should be under the control of the user.

3.1 Promised stability periods

The periodic adaptation of promised stability periods relates observations of past and present system conditions to determine a future stability period Δ_t.

Let $St = \{St_1, St_2, \ldots, St_n\}$ be the set of different types of services in the system. Each St_i may use different combinations (and amounts) of available resources R. Let $R_i = \{r_1, r_2, \ldots, r_n\}$ be the set of resources used by a task of a service of type St_i.

The periodic adaptation of promised stability periods for a task of a particular type of service is based on sampling the minimum period without QoS degradation of each resource $r_i \in R_i$, during a period of observation. Let $S_s = \{\Delta_{r_1}, \Delta_{r_2}, \ldots, \Delta_{r_n}\}$ denote those minimum stability periods of each resource of service St_i.

Any use of an arithmetic summarisation function that combines the values (such as a mean), will provide an incorrect stability period due to relative scaling. Relative scaling of n stability periods may lead to distortion when some values are particularly high or low. We need to process each value individually and offer a coherent summary to achieve a correct system behaviour.

Combination of several dynamical variables using AND (multiplication) and OR (addition) have already been discussed to provide more expressive policies for SLAs [8]. We propose to determine promised stability periods by aggregating the measured values for each resource used by a particular service type.

$$\Delta_t = \Delta_{r_1}\ AND\ \Delta_{r_2}\ AND\ \ldots\ AND\ \Delta_{r_n} \qquad (1)$$

Fuzzy logical reasoning provides a suitable interpretation of the stability periods of each resource in order to determine the promised stability period for a particular type of service. The use of the fuzzy AND operator (the min function) leads to a correct system behaviour and it is also computationally simple to evaluate. The promised stability period is thus given by

$$\Delta_t = min(\Delta_{r_i}) \in S_s \qquad (2)$$

3.2 Analysing traffic fluctuations

Effects of fluctuations in task traffic flow mean that there is an *uncertainty* associated with our estimation of the promised stability period for the next period of computation. If the flow is in a steady state we have a stronger degree of confidence to define needed resource usage for the next period than if the task traffic flow is showing larger fluctuations. An indication of the variability of the task traffic flow must be determined in order to obtain a confidence level $0 \leq c(St_i)_{k+1} \leq 1$ of resource usage for each service type St_i in the next observation period.

With services continuously arriving and departing the system it only makes sense to analyse short-term fluctuations in task traffic flow to try to predict the resource requirements in the next period of service execution. However, using only the current observation period seems to be too restrictive and very sensitive to strong variations in traffic flow. Using the previous and the current observation periods has the advantage of not punishing occasional strong variations as bad as systematic strong variations.

Let $l_{k-1} = \{l_{k-1}^{min}, l_{k-1}^{max}\}$ be a tuple with the minimum resource usage l_{k-1}^{min} and maximum resource usage l_{k-1}^{max} during the previous adaptation interval $k-1$.

Let $l_k = \{l_k^{min}, l_k^{max}\}$ be a tuple with the minimum resource usage l_k^{min} and maximum resource usage l_k^{max} during the current adaptation interval k.

Short-term fluctuations in task traffic flow are analysed by an intersection between the resource load variation of the current period of observation k with the load variation of the previous period $k-1$

$$I = min(l_{k-1}^{max}, l_k^{max}) - max(l_{k-1}^{min}, l_k^{min}) \qquad (3)$$

By assigning a meaningful value to this intersection it is possible to offer a confidence degree to the expected stability period in the interval $k+1$ for each resource $r_i \in R_i$. This confidence level is given by

$$c_{ri} = max(0, \frac{I}{L_k}) \qquad (4)$$

where $L_k = l_k^{max} - l_k^{min}$ is the length of variation in the current interval k.

When there is no intersection, I is negative and the confidence level is set to zero, indicating “no confidence” in the determined stability period, while a steady state in task traffic flow is rewarded with a confidence level of 1. This normalised value is a uniform and consistent way of representing confidence levels. Also, it is computationally simple.

The simplicity of computation of this intersection allows a quick evaluation of confidence levels on resource usage in the next period of computation.

$$c(St_i)_{k+1} = min(c_{r_i}) \in R_i \qquad (5)$$

4 Dynamic QoS reconfiguration

An undesirable high reconfiguration rate may be achieved by reconfiguring the offered SLAs on every task departure. Dynamic QoS reconfiguration should be based on a specific threshold of desired system utilisation. The goal is to reallocate needed resources to supply the initial SLA of each task that has suffered QoS degradation in order to accommodate new service requests.

Let L_t be the desired threshold to activate the dynamic QoS reconfiguration of previously degraded tasks whose stability period has already expired. Let L be the current level of system's load originated by the n currently existing SLAs. Intuitively, $L < L_t$ indicates an underutilisation of the service provider, which we want to avoid. When an underutilisation is detected the dynamic QoS reconfiguration will take place. The QoS optimisation problem involving multiple resources and multiple QoS dimensions is NP-hard [4]. Algorithm 1 implements a gradient descendent heuristic that starts with the initial contracted level for the previously downgraded tasks whose granted stability period has already expired and terminates when it finds a set of feasible QoS levels, if any.

A state change to a “better quality” is controlled by user's stability requirements specified in his request and a system's minimum confidence level of resource usage in the next period $c(St_i)_{k+1}$ for a particular type of service St_i.

Algorithm 1 Dynamic QoS reconfiguration

```
**Goal:** Minimise difference to initial SLA
Let $S_d$ be the set of previously degraded tasks whose $\Delta_t$ has expired
Let $S_o$ be the set of all other tasks
Let $Q_{kj}[i]$ be the current provided level for attribute $j$ of QoS dimension $k$ for task $T_i \in S_d$
Let $q[i]_u$ be the actual utility of each task $T_i \in S_d$

Select the initial SLA for all tasks in $S_d$, resulting in $S'_d$
while the new set of tasks $S = S'_d \cup S_o$ is not feasible do
  for each task $T_i \in S'_d$ receiving service at $Q_{kj}[m] > Q_{kj}[i]$ do
    Determine the utility decrease by degrading attribute $j$ to $m + 1$
    Find task $T_{min}$ whose reward decrease is minimum and degrade attribute $x$ to the $m + 1$'s level
  end for
end while
Let $q[i]'_u$ be the utility of each task in $S'_d$
for each task in $T_i \in S'_d$ do
  Let $St_i$ be the type of service of task $T_i$
  if $q[i]_u - q[i]'_u < U_{min}$ OR $\Delta_t(St_i) < \Delta_{min}(T_i)$ OR $c(St_i) < c(St_i)_{min}$ then
    Maintain the SLA for task $T_i$
  else
    Update the SLA of task $T_i$ to $S_d[i]'$
  end if
end for
```

Fluctuations in task traffic flow impose an uncertainty about needed resource usage for the next period of system's adaptation. Although the system always ensures that promised stability periods for each type of service are respected in the next period of adaptation, a confidence level on resource usage must be taken into consideration, associated with our estimation of the proposed stability period.

From Section 3.2, we know that a steady state of the task traffic flow has a stronger degree of confidence than large fluctuations in the flow. It is possible to define a more reactive or conservative system with respect to fluctuations in the task traffic flow by imposing a specific threshold of confidence on future resource usage that must be achieved in order to upgrade the quality of provided service of existing tasks. As the value of this confidence level approaches 1.0, it is harder to move to better quality states and stay in at least during the proposed stability period. This minimum confidence level for each service type $c(St_i)_{min}$ can also be dynamically adaptable (an issue for further study).

The new SLA proposal is evaluated according to user's preferences [5]. If the increment on service's utility achieved by this new SLA is greater than the user's imposed minimum increment U_{min}, the promised stability period for the next period is greater than the user's imposed minimum stability period Δ_{min}, and the system's minimum confidence level on resource usage for the next period St_i is achieved, then a service upgrade to the new SLA occurs.

5 Evaluation

A set of extensive experiments has been performed to verify the effectiveness of the proposed system's QoS adaptation decisions in response to fluctuations in task traffic flow. The following results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values used to drive the simulation [7]. The mean values of all generated samples were used to produce the charts.

At randomly generated times, one or more clients elaborated new service requests with random QoS constraints, for execution of one of the four different types of available services. Each type of service used a subset of the seven different system resources, with randomly generated capacities. These service requests originated the fluctuations in task traffic flow represented in Figure 1.

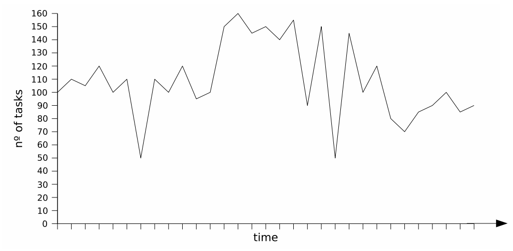

Figure 1. Fluctuations in task traffic flow

The observed average confidence level of resource usage for the next adaptation period was plotted in Figure 2. Notice that occasional strong variations on task traffic flow are effectively not punished as bad as systematic strong variations.

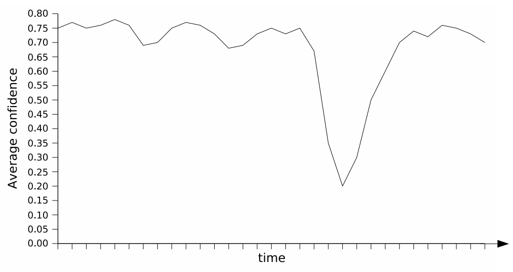

Figure 2. Average confidence levels

These confidence values were used to analyse the impact of setting different values for the system's minimum acceptable confidence level for an upgrade on the rate of QoS reconfigurations for a particular service type. Three service requests were generated for the execution of a task that has never left the system with the same spectrum of acceptable QoS values, in the same decreasing order of preference. The requests only differed on the QoS stability constraints for minimum utility increase and stability period, $U_1 = \{0, 0s\}$, $U_2 = \{0.2, 10s\}$, $U_3 = \{0.5, 30s\}$, respectively. The experience measured the reconfiguration rate experienced by each task, given by the number of changes to a better state divided by the period of observation. The results are presented in Figure 3.

Two important conclusions can be taken. First, analysing the reconfiguration rate of each individual user request, one can conclude that as the value of the minimum confidence level necessary to upgrade approaches 1.0, it is harder to move to a better quality state, which will only happen on a very stable task traffic flow. On the opposite side, a value close to 0.0 creates a more reactive system to fluctuations on task traffic flow that free enough resources to upgrade the SLA provided to existing tasks.

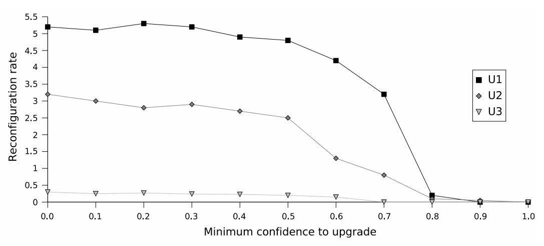

Figure 3. Impact of confidence levels

Second, the influence of personal constraints on the system's adaptation behaviour is clearly observable when comparing the three requests against each other. As the user's constraints for a service upgrade are harder to achieve there is less probability to change and stay in a better quality level.

6 Conclusions

Service providers promise users a specific level of service during a time interval Δ_t. Those periods are determined in response to fluctuations in task traffic flow, relating observations of past and present system conditions to promise a stability period for the future. We consider different stability periods for different types of services, since they may use different combinations (and amounts) of resources.

We believe that users should have some influence on how the system adapts its Service Level Agreements during applications' execution. Simulation results prove that such an influence can be achieved.

Furthermore, system's adaptation behaviour can be configured according to a desired confidence level on future resource usage, ranging from a very reactive system to fluctuations in task traffic flow to a more conservative system that only upgrades current SLAs in a steady flow.

Acknowledgements

This work was supported by FCT, through the CISTER Research Unit (FCT UI 608) and the Reflect project (POSI/EIA/60797/2004).

References

[1] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

[2] J. P. Hansen, J. Lehoczky, and R. Rajkumar. Optimization of quality of service in dynamic systems. In *Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems*, April 2001.

[3] B. Landfeldt, A. Seneviratne, and C. Diot. "user services assistant: An end-to-end reactive qos architecture. In *Proceedings of the 6th International Workshop on Quality of Service*, California, USA, 1998.

[4] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource qos problem. In *20th IEEE Real-Time Systems Symposium*, pages 315–326, 1999.

[5] L. Nogueira and L. M. Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, Colorado, April 2005.

[6] L. Nogueira and L. M. Pinho. Iterative refinement approach for qos-aware service configuration. In *Proceedings of the 5th IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006) (to appear)*, Braga,Portugal, October 2006.

[7] N. Pereira, E. Tovar, B. Batista, L. M. Pinho, and I. Broster. A few what-ifs on using statistical analysis of stochastic simulation runs to extract timeliness properties. In *Proceedings of the PARTES'04 Workshop*, Piza, Italy, 2004.

[8] G. Rodosek. Quality aspects in it service management. In *Proceedings of the 13th IFIP/IEEE Internation Workshop on Distributed Systems: Operations and Management*, pages 82–93, Montereal, Canada, October 2002.

[9] S. Schmidt, T. Legler, D. Schaller, and W. Lehner. Real-time scheduling for data stream management systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 167–176, 2005.