Dynamic Change Management for Minimal Impact on Dependability and Performance in Autonomic Service-Oriented Architectures

Tudor Dumitras, Daniela Rosu, Asit Dan, Priya Narasimhan

March 7, 2006
CMU-CyLab-06-003

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Dynamic Change Management for Minimal Impact on Dependability and Performance in Autonomic Service-Oriented Architectures

Tudor Dumitraş[1] tudor@cmu.edu
Daniela Roşu[2] drosu@us.ibm.com
Asit Dan[2] asit@us.ibm.com
Priya Narasimhan[1] priya@cs.cmu.edu

[1]ECE Department, Carnegie Mellon University, Pittsburgh, PA, USA
[2] Autonomic Service-Oriented Computing Group, IBM T.J. Watson Research Center, Hawthorne, NY, USA

Abstract

Dynamic change management in an autonomic, service-oriented infrastructure is likely to disrupt the critical services delivered by the infrastructure. Furthermore, change management must accommodate complex real-world systems, where dependability and performance objectives are managed across multiple distributed service components. This paper presents a change management framework that enables the assessment and minimization of service delivery disruptions. The framework builds on a few general principles. First, change management systems handle both external change requests, like software upgrades, and changes to mitigate internal events, like faults. Second, the impact on service delivery is assessed as the impact on the business values of the performance and dependability objectives across all services. The goal is to schedule change operations in order to maximize the business value across all service objectives over a long time horizon. These principles are achieved with a distributed design: change operation scheduling and business value optimization are performed by an orchestrator, while the impact assessment on specific objectives is distributed to objective-specific modules that employ domain-specific models for estimation and prediction.

1. Introduction

The Web based global economy of today demands not only a very high availability of online systems [2], but also satisfactory service levels (e.g., average response time), even when the system recovers from a failure, or when software upgrades occur in some parts of the system. Current change-management strategies, for the most part, tend to execute a change request as soon as possible (e.g. as soon as a fault has been detected or as soon as an upgrade has been requested). Enterprise IT infrastructures are customer-centric and business-driven, and the downtime (or the perceived lack of responsiveness) due to change management can often disrupt the performance expectations of services and have an adverse effect on business. Instead, it might be more appropriate to *seek the most opportune time to execute the change operations in a distributed infrastructure, based on the impact of the change on the service-level objectives* (e.g. response time, availability, and recovery time). Such an impact-sensitive change-management strategy would aim to respect the overall performance and dependability guarantees of the running services, yet allowing for the system to incorporate changes of various kinds.

Figure 1 illustrates the main elements of the change planning problem. First, in many IT infrastructures (e.g. supply chain management), there are multiple types and sources of change operations. These include external changes, derived from a request for change (RFC), and internal

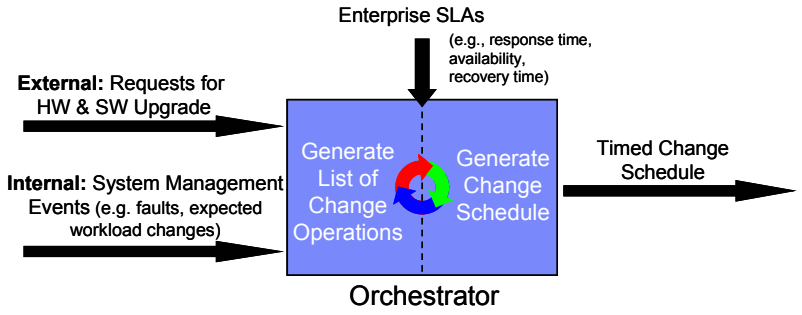

Figure 1. Dynamic change management problem

changes, triggered by system management events such as faults and workload surges. Change requests are characterized by a set of (partially) ordered change operations and by change objectives, such as the deadline for implementing the change and the associated penalties for failing to complete by deadline. The change operation planner must produce a timed schedule for executing the change operations, which considers the impact on all the relevant quality of service requirements, expressed by service-level objectives (SLOs), as well as the objectives of the change-operation. Each objective has a specific *business value* metric for evaluating the performability of a service and for gauging the utility of meeting the objective based on the level of service parameters, called Key Performance Indicators (KPIs) (see [6] for sample objective business value specification).

The planner must strive to maximize the aggregated *business value* of meeting all of the objectives during and after the change implementation. This optimization must be done over a long time horizon, to account for both transient effects, occurring during the change execution, and permanent effects, settling in after the change has been finalized.

The planner must evaluate the impact of change on service objectives by considering the inter-dependencies among various system components, the available knowledge of workload fluctuations or anticipated load surges during prime-time, as well as the degree of resource sharing between multiple services. However, it is difficult to perform such impact analysis in distributed enterprise infrastructures (e.g. supply-chain management), which usually integrate heterogeneous off-the-shelf components and sometimes span independent administrative domains. In these environments, the high-level service objectives translate into component-level objectives which are managed by component-specific configuration managers. For example, response time objectives of a service are managed by a workload manager that prioritizes and routes the service requests, while availability objectives are managed by a business resiliency manager that primes backup nodes in advance in anticipation of failures, and carries out the recovery process after a failure. These managers use extensive and sometimes proprietary domain knowledge (e.g., workload characteristics, resource utilization models) and can perform sophisticated

request classification, prioritization, monitoring and request routing; they may also use online learning or modeling to forecast arrival patterns and response times for certain resource configurations [3][9].

As a result, we submit that while in theory change management could be performed completely by a centralized, monolithic entity, the complexity and the distributed nature of objective management in real-world systems necessitates a decentralized framework for dynamic change management. Such a framework would delegate the assessment of change-operation impact on service objectives to the components that actually manage these objectives.

This paper proposes a distributed framework for change management that builds on this principle. A change operation orchestrator, responsible for building the schedule consults multiple objective advisors (e.g., performance advisor and dependability advisor) for assessing the impact of the change schedule on the service objectives. The advisors are software components that incorporate the domain knowledge (and sometimes specific product knowledge) to answer "what if" questions about service KPIs, such as recovery time and availability for the dependability advisor, given a description of the change operations and the timed schedule for executing them. The orchestrator asks these "what if" questions using a Web Services-based protocol, and then uses the returned service KPI predictions to compute the aggregated business value and to converge toward the optimal schedule through an iterative refinement process. The objective advisors themselves can be composite services and they can be provided by third-party vendors. Unlike a centralized approach to change management, our proposal facilitates the runtime integration of new off-the-shelf components.

The main contribution of this paper is a novel distributed framework for orchestrating change operations in service-oriented architectures, which has the following characteristics:

- Accounts for heterogeneous types and sources of change operations: internal (e.g., faults, workload changes) and external (e.g., request for change);
- Assesses the impact on the enterprise SLOs, as well as the change request deadlines;
- Evaluates the variation of performance and dependability metrics over a long time horizon, provided by domain-specific knowledge of multiple service management components;
- Enables the optimization of long-term business value, both during the change execution, and after the change has been finalized.

We describe the details of this framework, which focuses on the interaction protocol between the distributed components, and we present real examples arising from software upgrades and shifting workloads as part of fault recovery. We would like to note that the problem formulation, scenarios and

many of our observations are drawn from real world examples and knowledge of existing commercial systems managing such environments.

This paper is organized as follows: Section 2 reviews the relevant related work in the area of dynamic change management. Section 3 describes concrete change management scenarios that emphasize the need for impact assessment. Section 4 describes the distributed framework for dynamic change orchestration. Section 5 discusses our results and outlines directions for future work. We conclude by summarizing our main ideas.

2. Related work

In their excellent survey, Segal and Frieder [16] identify a set of general requirements for any dynamic updating system: preserving program correctness (during and after the update), minimizing human intervention, supporting program restructuring and low-level program changes (e.g., both implementations and interfaces), supporting distributed programs (communicating across mutually distrustful administrative domains), not requiring special-purpose hardware and not constraining the language and environment. The survey illustrates that in general, research has focused on mechanisms for implementing change at different levels of granularity (e.g. replacing components, objects, procedures), rather than on impact assessment and coordination of distributed changes.

Kramer and Magee [11] note that faults, as well as live upgrades, might have a disruptive effect on the functionality of a distributed system, and that the techniques to mitigate these problems could be combined in a unified framework. For instance, a change-management system that totally separates the functional application concerns from the configuration management concerns (such as Kramer and Magee's Conic system), can provide a good basis for implementing fault recovery [11]. Conversely, an infrastructure built for fault-tolerance can provide a good basis for live upgrades because of the inherent redundancy [1][12]. For example, a fault-tolerant CORBA system using the interception approach provides all the ingredients needed for dynamic change management of CORBA objects, including an interceptor (i.e., the indirection layer needed when switching to a new version), replication mechanisms (for incrementally upgrading some replicas while others continue to provide service) and state extraction/restoration mechanisms (for maintaining consistency between versions) [12]. In our framework, we also adopt this unifying approach of considering both external (e.g. software upgrades) and internal change requests (e.g. operations needed to mitigate the impact of a fault).

Some existing change management products, such as the IBM Tivoli Intelligent Orchestrator (TIO) [8], perform resource arbitration between node groups by evaluating the immediate impact of the resource changes. While allowing a distributed management of the enterprise infrastructure [3], this approach is limited because it ignores the long-term impact of change management (e.g. interaction with

expected workload change). The CHAMPS project [10] focuses on scheduling operations to satisfy external RFC time objectives. It develops a complex dependency-tracking framework and it formulates the scheduling problem as the optimization problem given a set of execution constraints and a generic cost function that models the impact during change operations. Complementing CHAMPS' centralized approach to cost evaluation, we propose a framework and a protocol for distributed cost evaluation based on component-specific assessment of long-term impact of change operations on service KPIs.

Gorbenko et al. [7] tackle the problem of achieving high dependability of composite Web Services undergoing online upgrades of their components. They advocate running multiple versions of a service in parallel and using third-party interception middleware to switch to a new replica when the confidence in its correctness is sufficiently high. The "confidence in correctness" metric is computed based on comparing the responses from different versions of a service and using Bayesian inference to reason about future failure rates. This approach is the closest to our focus on the long-term impact of change operations, except that we use impact assessment across multiple service-level objectives and we use standard metrics, such as business value, for evaluating this impact.

3. Dynamic change management in service-oriented architectures

To illustrate the problem more concretely, we present two realistic change management scenarios in a complex, but not necessarily very large, enterprise infrastructure. We consider a two-tiered system, where the physical hosts are organized in node groups that are managed by independent software components. The first tier is a node group of application servers servicing Web transactions and managed by application server middleware (e.g., IBM WebSphere Extended Deployment [9]) and the second tier is a node group of database servers, managed by database cluster infrastructure (e.g., Oracle Clusterware [13]) (see Figure 2). The two management components perform various middleware specific management tasks (e.g., load balancing, request routing, replication, fault recovery).

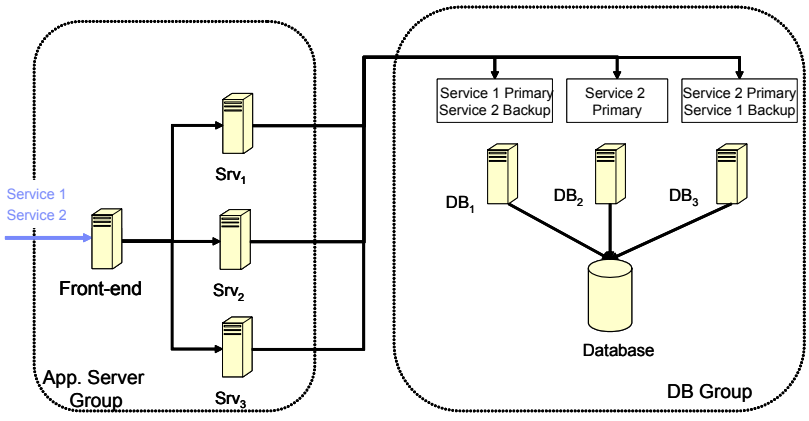

Figure 2. Example: two-tier system

Table 1. Service objectives in sample enterprise infrastructure

Objective	Parameters	Sample Penalty Function
Average Response Time	Target avg, response time, averaging interval (e.g., 5 min.)	0.20 x max(0, TotalTrans/TransOnTarget – 1)
Recovery Time	Target recovery time	2000 x max(0,ExpectedTime/ TargetTime -1)
Availability	Target availability percentage, evaluation time interval (e.g., one month)	200 x max (0, ActualAvail– TargetAvail)

This infrastructure provides two services, each mapped on corresponding application-server and database services. The two application server services are load-balanced across the three servers, Srv_1 to Srv_3. The database services connect to two separate database partitions. The database group comprises three servers:

- DB_1 acts as primary server for `Service1` and as backup for `Service2`;
- DB_2 is part of the logical primary server for `Service2`, which is distributed on two database nodes;
- DB_3 is also part of the logical primary for `Service2` and it is a backup for `Service1` as well.

Each of the two enterprise services is subject to response time, recovery and availability objectives (see Table 1). The business value associated with these objectives is expressed as penalty functions, depending on related service KPIs, such as 'total number of transactions', 'number of transactions with response time below target', etc.

In this context, the goal of the change management system is to create change schedules that maximize the aggregate business value across all objectives during and after change implementation.

3.1. Scenario 1: node-fault management

A sample change management scenario is related to managing recovery after hardware faults. When a fault is detected in one of the nodes, the corresponding node-group manager takes immediate recovery measures. For example, when DB_1 suffers a hardware crash, the database recovery manager handles the failover of `Service1` to its backup node, DB_3. As a result, DB_3 handles queries for both services, while DB_2 continues to handle only queries for `Service2`. However, since the database group now has fewer nodes, thus higher risk of failing the availability objectives, the change management system must decide whether removing one node from the application server group and adding it to the database group would improve the overall business value and when these operations should be scheduled.

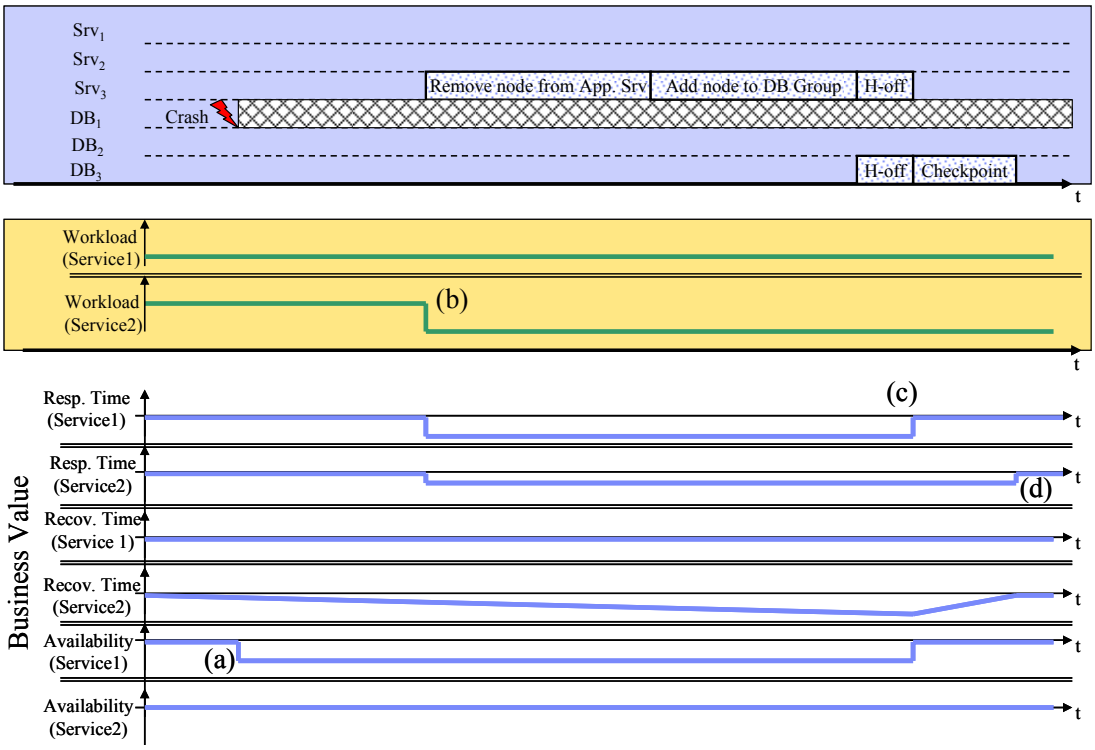

Figure 3. Impact of change operations for sample scenario

Figure 3 shows the impact of these change operations. Namely, after the crash of DB_1, the lack of a backup leads to a sharp decrease of the predicted availability of `Service1` (the mean time to failure is shorter) and a drop in the corresponding business value (i.e., value in negative range due to non-zero penalty) – indicated by point (a) in the figure. However, since the load of `Service2` is high at this point, transferring a node from the application-server group to the database group would lead to a severe penalty due to failing to meet the response time objective. Therefore, the change operations should be delayed until the load of `Service2` decreases, at point (b). During the node transfer, the response time decreases for both services, but after the hand-off – point (c) – the response times, as well as the availability of `Service1`, may return to normal (i.e. zero penalty). However, since `Service2` has been continuously sending queries to the database, its log kept growing, leading to an increase of the recovery time. To solve this problem, a database checkpoint[1] is requested and it must be carefully scheduled not to conflict with the other change operations. After the end of the checkpoint, indicated by point (d), the response time and the recovery time for `Service2` decrease to normal operating levels.

[1] A database checkpoint is an operation that synchronizes the modified data blocks in memory with the data files on disk, thus shortening the portion of the log that needs to be processed during a recovery operation.

3.2. Scenario 2: database software upgrade

A similar impact analysis must be undertaken when upgrading the database software. In this case, a request for change is decomposed into finer grain change operations: each database node is upgraded separately and, for upgrading DB_1, `Service1` is handed off to DB_3 (its backup) before the upgrade and restored at the end. The analysis must consider the impact of these operations on service objectives and their corresponding business values (see Table 1). For instance, if the load on `Service1` is high, we can reorder the change operations to perform the upgrades on nodes DB_2 and DB_3, which are used by `Service2`. In fact, the upgrade of DB_1 must be delayed until both services register low incoming request rates because a high request rate during the upgrade may overload DB_3, which also handles both `Service1` and `Service2`. By delaying the upgrade, the penalties incurred for violating the response time objectives are minimal, thus maximizing the aggregate business value for the duration of the changes. The reordering must take into account the dependencies between change operations; thus, the hand-offs of `Service1` should precede and follow the upgrade of DB_1.

These two scenarios are typical for the change management in an enterprise infrastructure; similar operations occur at a much larger scale in many real-life deployments. Using a traditional scheduler, which does not optimize for long-term impact [8][10], would result in executing all of the change operations as soon as possible, instead of waiting for the most opportune time when the incoming load is low. Such impact-insensitive scheduling would thus result in a missed opportunity for optimizing the overall business value. Also, these scenarios illustrate the complexity of predicting the impact of change on service objectives due to the strong dependencies on the actual implementations of objective managers. This emphasizes the need of a change management framework that enables the optimization of change plans based on the long-term impact on service objectives assessed by objective-specific managers.

4. Distributed framework for generic change management

The main design goal for a change management framework that targets distributed, service-oriented infrastructures is to make minimal assumptions about the kinds of knobs that the various software components are prepared to expose to the change management system. The key to achieving this goal is the separation of scheduling and impact analysis.

More specifically, in our framework (see Figure 4), a centralized component, called *orchestrator* receives the requests for change, identifies the component operations and their execution constraints, and generates a scheduling plan in an iterative process. The impact analysis of scheduling plans is performed by distributed components called *objective advisors*, which represent the service managers in the infrastructure, and can use manager-specific knowledge to estimate the impact of a plan on the managed

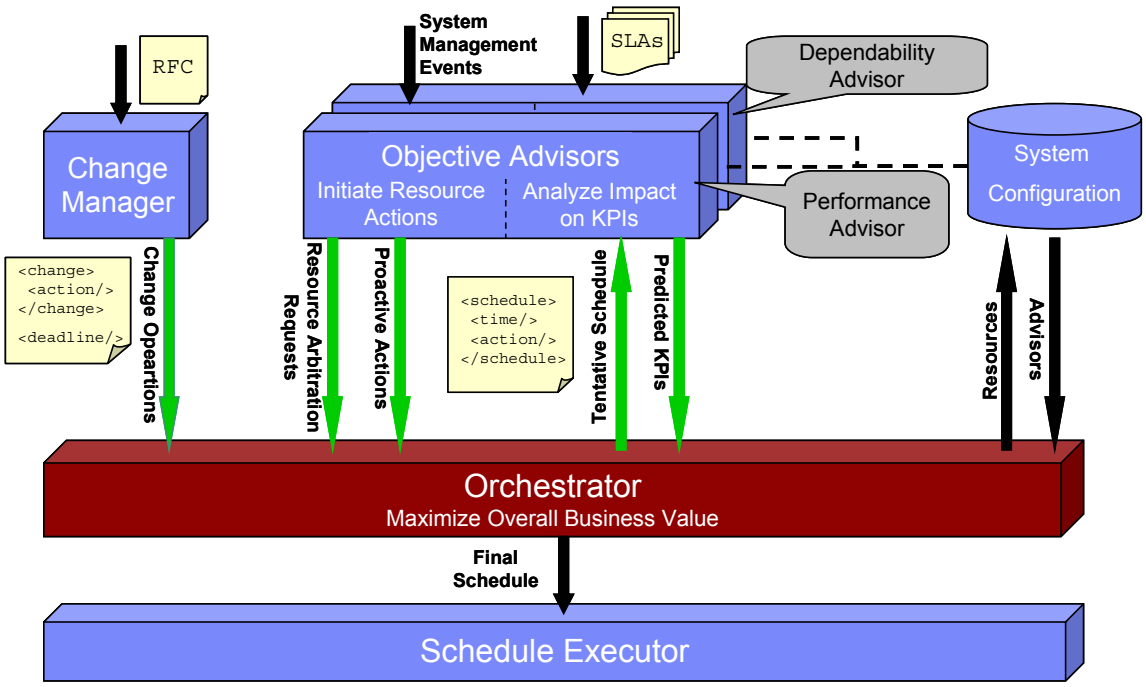

Figure 4. Distributed architecture for change management

service KPIs. The orchestrator consumes these estimations and schedules the change operations with the objective of maximizing the business value. The interaction between the orchestrator and advisors is based on specialized APIs. The communication is based on the Web Services standard, which guarantees compatibility in a complex system with components coming from different providers.

We assume that the objective advisors are able to predict future incoming loads (as illustrated in Figure 3), either because the workloads have a strong periodicity [4], or because fluctuations are preceded by recognizable patterns of warnings and notifications [14][17]. Furthermore, we assume that the execution times of all the change events submitted to the orchestrator can be estimated with a certain precision and that the running services do not have hard real-time constraints (this is typically the case in enterprise infrastructures).

In the following, Section 4.1 presents details about the framework components, Section 4.2 presents the interaction protocol for impact assessment, Section 4.3 presents the business value model used for scheduling plan optimization and Section 4.4 presents the scheduling algorithm.

4.1. Framework components

Figure 4 illustrates the main components and interactions in the proposed framework. The high-level RFCs are received by the Change Manager, which decomposes them into finer grain change operations and related dependencies, and forwards them to the Orchestrator. The Orchestrator decides the timed schedule of the operations by interacting with the objective advisors and by exploiting information from

System Configuration Database regarding system resources and related objective advisors. The timed schedule is provided to the Schedule Executor, which triggers the change operations at the indicated times. The Change Manager is analogous to the Task Graph Builder from [10], and the Schedule Executor is similar to TIO Provisioning Manager [8]. In the following, we focus on the orchestrator and objective advisors, which are novel to our approach.

Orchestrator. The orchestrator is a resource broker and a planner of change operations. The orchestrator is invoked in one of three situations: (i) when a change sequence has been initiated, following a request for change; (ii) when a predicted or observed infrastructure event (e.g., a fault, a workload change) mandates a resource reassignment; and (iii) when a service-level agreement has changed, indicating a potential change in overall business value expression. The output of the orchestrator is a timed schedule of when the change operations should be initiated. All the invocations of the orchestrator are asynchronous (i.e. a response containing the schedule is not provided immediately).

The API of the orchestrator contains three functions:

- `InitiateChange()`: request for scheduling of a group of change operations, such as one derived from an RFC. The orchestrator computes a schedule for implementing the operations.
- `InitiateResourceBrokering()`: request for reallocation of resources (e.g. nodes) to mitigate the impact of an event detected by the system management infrastructure (e.g. a hardware fault). The orchestrator determines a resource allocation that improves the overall business value, and if any, it computes a schedule for implementing the corresponding change operations.
- `ChangeSLA()`: request for integration of SLA updates. Orchestrator retrieves the descriptors of objective business value expressions for use in its scheduling decisions (more comprehensive mechanisms for managing SLAs updates are described in [5]).

During the scheduling process, the orchestrator communicates with the objective advisors in order to assess the impact of tentative change operation schedules on the future service KPI values. Based on the predicted KPIs, the orchestrator computes the objective business values using the models defined by the SLAs in place. The objective business values are aggregated into the overall utility of the schedule. Using this metric to compare different schedules, the orchestrator converges, through an iterative process, to the best feasible schedule.

Objective advisors. The objective advisors are associated with service managers. They can be hierarchical or they can span multiple managers in order to manage end-to-end KPIs. The advisors estimate the impact of observed, predicted or scheduled events on the service KPIs managed by the

associated managers. The advisor invocations are synchronous (i.e. KPI predictions are provided in response to these invocations).

The API of an advisor contains two functions:

- `GetCurrentKPIs()`: request for current KPI predictions for a given time interval, assuming that only infrastructure events (e.g., workload variation, node failures) will occur. The reply references each of the KPIs handled by the advisor.
- `GetImpactKPIs()`: request for KPI predictions over a given time interval given a schedule of change operations. The reply can include a set of *proactive actions* (e.g., checkpoint database) expected to improve the KPIs in conjunction with the change operations; for each set of proactive actions, the reply includes a full set of KPI predictions. This function allows the orchestrator to evaluate "what if" scenarios for assessing the impact of planned events on the enterprise objectives.

For KPI monitoring and prediction, the advisors exploit the management APIs provided by the vendors of the management component software. They do not depend on the actual enterprise business value models, which are handled by the orchestrator.

Sample objective advisors include performance and dependability objective advisors. In the enterprise scenario described in Section 3, a performance advisor evaluates the impact of change operations on the end-to-end response time for each service by exploiting the knowledge provided by the application server and the database node group managers (e.g., expected workload variations, service overheads). Similarly, the dependability advisor evaluates the impact on the recovery time and the availability KPIs. For the evaluation of our approach, we have developed configurable emulators for these two advisors.

4.2. Interaction protocol for impact assessment

The interaction protocol is at the heart of our approach. As shown in Figure 4, the orchestrator starts making a change plan after the ChangeManager invokes the `InitiateChange()` function or the advisors invoke the `InitiateResourceBrokering()` function. The orchestrator calls the `GetCurrentKPIs()` and `GetImpactKPIs()` functions of each of the advisors, creating and refining a schedule through an incremental process. The advisors provide KPI predictions in response to suggested schedules and they may also annotate the schedules with proactive actions, that the orchestrator may include in the schedule in order to alleviate the impact on the KPIs. These proactive actions (e.g. the checkpoint in the example from Section 3.1) correspond to a certain type of resource (e.g. database) or change operation (e.g. node transfer) and they are included in the final schedule only if they improve the overall business value.

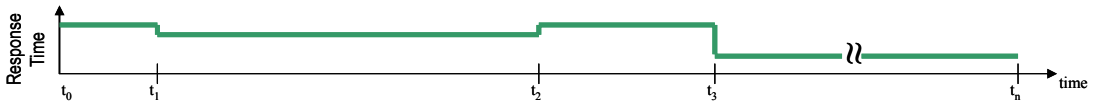

Figure 5. Long-term KPI variation

For minimizing the communication costs, the interaction protocol provides for locally caching business value information related to partial schedules. Each partial schedule receives a unique identifier, known to the orchestrator and advisors, and the related KPI predictions are saved. The KPI predictions are recalled whenever the partial schedule is modified by adding one or more operations, thus avoiding the repetition of most of the computations.

4.3. Business value model

The service objective business value (BV) is a function that associates a dollar value with various levels of service provided by the system. As service levels are defined by one or more service KPI values, a business value function depends on one or more KPI parameters. For simplification, we assume, a business value function depends on only one KPI. At time *t*, the KPI value is *KPI(t)* and the corresponding business value is $BV_{Obj}(KPI(t))$. A KPI value is assumed to hold for a period of time, until an infrastructure event (e.g., a fault or a workload surge) or a change event (e.g., a SW upgrade or a resource configuration change) causes the KPI to take another value (see Figure 5). The business value functions that correspond to different KPIs are assumed to be additive. The instant business value for a state of the system is: $BV_{All}(t) = \sum_{\text{All } Obj_k} BV_{Obj_k}\left(KPI_k(t)\right)$. For each KPI that changes at times t_0, t_1,… t_n, the business value for the time interval $[t_0, t_n]$ is computed using a weighted average:

$$BV_{Obj}([t_0, t_n]) = \frac{\sum_{i=0}^{n-1} BV_{Obj}(KPI(t_i))(t_{i+1} - t_i)}{t_n - t_0}$$

The business values for all the KPI are then added to compute the overall business value of the schedule of events:

$$BV_{All}([t_0, t_n]) = \sum_{\text{All } Obj_k} BV_{Obj_k}([t_0, t_n])$$

4.4. Scheduling algorithms

The scheduler is a sub-component of the orchestrator that produces timed schedules for change operation groups. A schedule indicates when each individual operation from the group should start executing. The goal of the scheduler is to maximize, for a certain time horizon, the overall business value across service objectives. A group of change operations corresponds to a request for change (RFC) or to a request for

resource brokering.

Each group comprises one or more change operations and defines a partial order relation between the operations, indicating their precedence dependencies. Furthermore, a group may specify a deadline for completing the execution of all operations and a business value expression that reflects the penalty of late completion (similar to the other system-level objectives from Table 1). This business value is factored into the overall business value of the system to be maximized by the scheduler. The scheduler's time horizon for business value evaluation must be long enough to include the deadline. If the deadline is missing, then the aggregated business value of the service-level objectives is the only criterion for selecting a schedule. A change operation group also has a priority value associated, which indicates its criticality relative to other groups pending for scheduling. Upon the arrival of a high priority group, the scheduler can preempt its ongoing processing of a lower priority group and start processing the new request; the processing of the preempted group will be restarted at a later time.

Operations in a group can be mandatory, such the operations derived from an RFC, or optional, such as operations of node reallocation proposed by the orchestrator in response to resource brokering requests. The scheduler can discard optional operations if they do not improve the overall business value. The set of operations in a group can expand during the scheduling decision due to the proactive actions suggested by the objective advisors in response to a proposed schedule (e.g. checkpoint database from Section 3.1). In general, the proactive actions are optional.

A change operation is defined by a name, a scope and a set of properties. The name is an enterprise-specific descriptor recognized by all of the related objective advisors and service managers, e.g., "Upgrade database software to version 10.0". The scope identifies the resource(s) involved by the operation, e.g., "database node DB_1". The properties are a list of <name, value> pairs that describe operation characteristics such as the duration of executing the operation, the additional load (e.g. in terms of CPU% and memory) that it will impose on the host, etc. Some properties are defined in the change request and others can be retrieved from the System Configuration database.

Another input to the scheduler is the list of KPI predictions over the scheduling time horizon. The scheduler starts by invoking the `GetCurrentKPIs()` function of the objective advisors to retrieve the future variation of all the relevant KPIs due to infrastructure events (e.g., faults, workload surges). These predictions indicate the time instants when the KPIs will be changing (see Figure 5), which divide the time horizon into intervals that can be used as scheduling guidelines. Henceforth, these time instants are called '*prediction points*'.

The scheduler's goal is to associate start times $t_1, t_2 \ldots t_n$ with the operations $e_1, e_2 \ldots e_n$ in a change-

operation group such that it complies with the partial ordering among operations and the group deadline (if defined). In this process, the scheduler generates a series of partial or complete operation schedules. For each schedule, it queries the objective advisors for predictions of the schedule impact on KPIs during the scheduling time horizon; these predictions are used to compute the overall business value of the schedule by aggregating all of the objective business values and the group deadline business value, if defined. The final schedule generated by this algorithm should provide the best possible business value.

Complexity. The scheduler does not know the general form of the function that yields the overall business value because part of this computation is performed inside the objective advisors, which act as black boxes for the orchestrator. In scheduling-theory terms, this means that the scheduling problem has an unknown objective function [15]. Given that the complexity of scheduling algorithms depends on their objective functions, it is impossible for us to reason about the complexity of our problem. Moreover, even if we had a closed-form expression for the business value, this would most likely be a non-regular objective function (i.e. would not be non-decreasing in the completion times of the change operations); there are few theoretical results for scheduling problems with such objective functions.

Scheduling assumptions. For this study, we make few simplifying assumptions for designing scheduling algorithms for this problem. First, we assume that all operations in a change group are mandatory. Second, we assume that all the change-operation groups have explicit deadlines. When not defined explicitly, we can set the deadline at the end of the time horizon for business-value evaluation; it makes no sense to schedule operations past this time horizon because we would not be able to see their impact on the business value. Third, the operations in a change group are totally ordered (i.e. an operation must complete before the next one can begin). This is the most constraining assumption, since it precludes the re-ordering of operations inside a group (as we have shown in the software upgrade scenario from Section 3.2).

Scheduling algorithms. Table 2 synthesizes the basic input and output parameters for the scheduling algorithm. There are n change operations to be scheduled and m prediction points serving as scheduling guidelines. If the change group is not schedulable, no output is provided; this happens when all the change operations cannot be completed before the deadline: $\sum_{i=1}^{n} d_i > D$.

The algorithms considered in this study are based on the following pattern. For each operation e_k, we compute the earliest and the latest possible times when it can be scheduled:

$$\sum_{i=1}^{k-1} d_i \le t_k \le D - \sum_{i=k}^{n} d_i$$

Table 2. Input and output parameters for the scheduling problem

Input Parameter	**Notation**
Set of change operations	$\{e_1, e_2 \dots e_n\}$
Set of durations of corresponding change operations	$\{d_1, d_2 \dots d_n\}$
Set of prediction points	$\{Pp_1, Pp_2 \dots Pp_m\}$
Scheduling deadline	D
Output Parameter	**Notation**
Schedule of change operations	$\{t_1, t_2 \dots t_n\}$

Using these bounds, we try to schedule each operation at its earliest time, its latest time and at all the prediction points that fall within this feasible interval.

The simplest algorithm uses backtracking to test all the possible combinations. In precedence order, we place each operation at each of the prediction points in its feasible interval and then, for each of these placements, we repeat the algorithm for the remaining operations. If the KPIs are expressed as step functions, as shown in Figure 5, and the business values are linear functions of the KPI values (which would make them step functions as well), this algorithm generates the optimal schedule. Because we try to place an event at all the prediction points in the feasible interval and then, for each of these placements, we repeat the same algorithm for the remaining events, this algorithm, henceforth called `Backtracking`, has the worst-case complexity $O(m^n)$.

Another approach is to use a polynomial best-effort algorithm that is not guaranteed to provide an optimal solution. We can achieve this with a greedy algorithm: we place each operation at each possible position and select the operation and the placement that yield the best possible business value. This placement splits the timeline and the change-operation group in two and the same algorithm is applied for the two segments of the problem (see Figure 6). This algorithm, henceforth called `Greedy1` has the complexity $O(n^2m)$ because, for scheduling each of the n operations, the algorithm evaluates nm placement options..

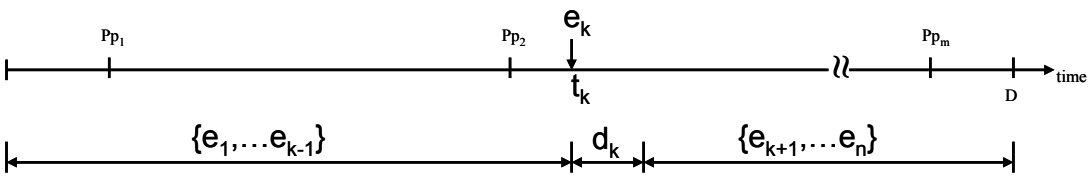

Figure 6. Greedy scheduling algorithm

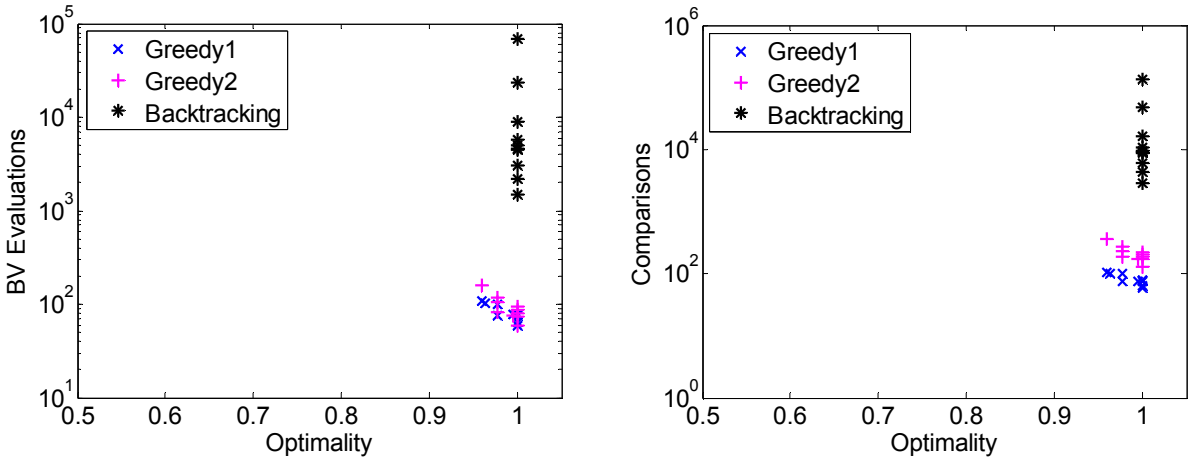

Figure 7. Cost vs. loss of optimality trade-off

This algorithm has the disadvantage that it tends to give priority to the short operations that have a small negative impact. These operations get the best placements, sometimes leaving the large operations to be scheduled during busier periods, thus affecting the overall business value. To avoid this situation, we can modify the selection condition to choose the operation that displays the largest business value variation depending on the scheduling time. This strategy leads to selecting the operation most sensitive to placement first. This algorithm, called `Greedy2`, has the same complexity as the previous one.

We test these algorithms using randomly-generated input sets. For small problems (e.g. 5 change operations and 10 prediction points), we can explore the trade-off between two measures of complexity and the loss of optimality. The complexity measures are: the number of times the business value needs to be evaluated and the number of comparisons. The loss of optimality is measured as the ratio of the BV of the resulting schedule and the BV of the optimal schedule, generated by the backtracking algorithm. Figure 7 shows that the two (polynomial) greedy algorithms obtain near-optimal results, while the number of operations they require is one or two orders of magnitude lower than in the case of the exponential, optimal backtracking algorithm. Furthermore, the two quadratic algorithms seem to have sensibly different costs: `Greedy2` makes more than twice as many comparisons as `Greedy1` because it needs to keep track of both the highest and the lowest business value.

For larger problem sets, we cannot run the backtracking algorithm, and therefore we cannot measure the loss of optimality of the other two schedulers. For 100 change events and 100 prediction points, the greedy algorithms required up to 36673 business value evaluations and 67342 comparisons, sometimes with significant differences between the two algorithms (between 3% and 68%). `Greedy1` also exhibits a higher variance of the number of BV evaluations than `Greedy2`. While we could easily construct a

scenario where `Greedy2` would perform better than `Greedy1`, the two algorithms produced identical schedules for all but one of the randomly generated scenarios.

5. Discussion

By focusing on the communication protocol for impact assessment rather than on building a monolithic change management system, we facilitate changes that may span multiple independent administrative domains and that may target uncooperative software infrastructures. Using the protocol described in this paper, a generic orchestrator can communicate with third-party advisors, built with specific, proprietary domain knowledge about a service/system/provider and construct schedules using only the information that is available from such advisors. This makes our approach widely applicable, although it may limit the optimization capabilities when the advisors cannot provide a comprehensive impact analysis (e.g., some services may not provide latency impact analysis required for end-to-end response time management). Appropriate orchestrator implementations can generate change schedules even with imperfect information or predictions about the system; however, the quality of the schedules will improve significantly if accurate impact analysis is available. Furthermore, the business value computation method presented in this paper can be easily adjusted to take into account statistical KPI or workload predictions, such as confidence intervals or decreasing weights for the estimations that are distant in future (which tend to be less accurate). Our approach mirrors the philosophy of the Service-Oriented Architectures, which is to focus on interaction protocols, rather than implementation bindings.

Since the previous work in this area did not focus on analyzing the impact of change management on the online services, we could not find any more example scenarios to help us benchmark our framework. One open question is the typical size of a realistic change-operation group, which is important for selecting a good scheduling algorithm. If such a group (which corresponds to a single RFC document or to a set of recovery operations related to an infrastructure event) is usually small, then the deterministic algorithms such as the ones described in this paper might be good enough. For larger problem sets we plan to investigate heuristics such as genetic algorithms or simulated annealing [15].

Another issue that warrants further exploration is the best way to express the KPI variation in time. The step function representation used in this paper might be too constraining; for instance, it cannot describe a recovery time that increases linearly with the increase over time of the database log, as depicted in Figure 3. However, this representation is easy to understand and to use (as opposed to a describing a generalized function), and it can approximate well an arbitrary KPI trajectory if enough change points are selected. Furthermore, using the step function change points as scheduling guidelines, enables simple algorithms even for a scheduling problem with an unknown objective function,

For the future, we plan to integrate the framework components described in this paper with a real system in order to study in a realistic setup the impact of faults and upgrades, as well as the amount of prediction and impact analysis that we can perform in order to support an impact-sensitive change management planner. We also plan to experiment with various workload and failure scenarios, environment configurations as well as with various service-level objectives for performance, availability and details of requests for change.

6. Conclusions

This paper investigates the problem of performing dynamic change management while maximizing the aggregate business value across all service level objectives of the enterprise. We illustrate this problem using realistic scenarios and we show that impact assessment is essential for maximizing the business value. We propose a novel framework for distributed implementation of change management by separating the impact assessment (performed by the goal advisors) and the scheduling and business value aggregation (performed by the orchestrator). This approach, centered around a communication protocol rather than on implementation bindings, delegates the assessment of change-operation impact on service objectives to the objective-manager components in order to leverage their embedded domain-specific knowledge. This also allows us to coordinate changes that span multiple administrative domains and heterogeneous services and software components. The framework takes into account the impact of change management on the enterprise SLOs, the long-term KPI variation and heterogeneous types and sources of change operations (both internal and external). We present three deterministic scheduling algorithms and we compare experimentally the trade-off between their cost and their loss of optimality.

Acknowledgments

The authors would like to thank Biswaranjan Bhattacharjee and Joel Wolf of IBM Research, Florin Oprea of Carnegie Mellon University and Jean-Charles Fabre of LAAS CNRS for their invaluable input in the early stages of this research.

References

[1] T. Bloom and M. Day, “Reconfiguration in Argus,” *International Workshop on Configurable Distributed Systems*, Mar 1992, pp. 176-187.

[2] E. Brewer, “Lessons from Giant-Scale Services,” *IEEE Internet Computing* 5(4), Jul. 2001, pp. 46-55.

[3] D. Chess, G. Pacifici, M. Spreitzer, M. Steinder, A. Tantawi, and I. N. Whalley, "Experience with Collaborating Managers: Node Group Manager and Provisioning Manager," *Second International Conference on Autonomic Computing (ICAC'05),* 2005, pp. 39-50.

[4] J. Dilley, “Web server workload characterization”. *Technical Report HPL-96-160*, Hewlett-Packard Laboratories, Dec.

1996.

[5] D. Roşu and A. Dan, "Managing end-to-end lifecycle of global service policies," *International Conference on Service Oriented Computing (ICSOC)*, 2005.

[6] Global Grid Forum, *Web services agreement specification (WS-Agreement)*. Draft, version 11, 2004.

[7] A. Gorbenko, V. Kharchenko, P. Popov, and A. Romanovsky, "Dependable Composite Web Services with Components Upgraded Online", *Architecting Dependable Systems Vol. III (LNCS 3549*), edited by C. Gacek, A. Romanovsky and R. de Lemos, Springer-Verlag, Sep 2005, pp. 92 - 121

[8] IBM Tivoli Intelligent Orchestrator. http://www-306.ibm.com/software/tivoli/products/intell-orch.

[9] IBM WebSphere Extended Deployment, http://www-306.ibm.com/software/webservers/appserv/extend.

[10] A. Keller, J. Hellerstein, J.L. Wolf, K. Wu and V. Krishnan, "The CHAMPS System: Change Management with Planning and Scheduling", *Network Operations and Management Symposium (NOMS 2004)*, April 2004

[11] J. Kramer, J. Magee and A. Young, "Towards unifying Fault and Change Management," *International Workshop on Distributed Computing Systems in the '90s*, Cairo, Egypt, 1990, pp. 57 - 63.

[12] L.E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. Tewksbury and V. Kalogeraki, "Eternal: Fault Tolerance and Live Upgrades for Distributed Object Systems", *DISCEX Information Survivability Conference*, Hilton Head, SC, January 2000, pp. 184-196.

[13] Oracle Corporation, "Oracle Real Application Cluster 10g", *Oracle Technical White Paper*, May 2005.

[14] S. Pertet and P. Narasimhan, "Proactive Recovery in Distributed CORBA Applications," *IEEE Conference on Dependable Systems and Networks (DSN)*, Florence, Italy, June 2004, pp. 357-366.

[15] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. 2nd edition, Prentice Hall, 2002

[16] M. E. Segal and O. Frieder, "On-the-Fly Program Modification: Systems for Dynamic Updating," *IEEE Software,* 10(2), Mar. 1993, pp. 53-65.

[17] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo. "Workload-Aware Load Balancing for Clustered Web Servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 3, Mar. 2005, pp. 219-233.