RC24125 (W0611-194) November 29, 2006
Computer Science

IBM Research Report

Impact-Sensitive Framework for Dynamic Change-Management

Tudor Dumitras*, Daniela Rosu, Asit Dan, Priya Narasimhan*
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

*Carnegie Mellon University

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Impact-Sensitive Framework for Dynamic Change-Management

Tudor Dumitraş
Carnegie Mellon University
tudor@cmu.edu

Daniela Roşu
IBM Research
drosu@us.ibm.com

Asit Dan
IBM Research
asit@us.ibm.com

Priya Narasimhan
Carnegie Mellon University
priya@cs.cmu.edu

Abstract

This paper presents a distributed change management framework that enables the assessment and minimization of service-delivery disruptions. The framework handles both external change requests, like software upgrades, and changes to mitigate internal events, such as faults. Objective-specific modules assess the impact of change operations on service delivery through their impact on the business values of the corresponding performance or dependability objectives. A centralized component then schedules the changes in order to maximize the business value across all service objectives over a long time-horizon. We evaluate this framework using a realistic change-management scenario.

1. Introduction

Enterprises demand highly available online systems and satisfactory service levels (e.g., average response time) in the face of faults and upgrades. 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 is detected or an upgrade is requested). The downtime (or the perceived lack of responsiveness) due to change management can disrupt the performance expectations of services and have an adverse effect on business. Gartner Group reports that 80% of application-service downtime is directly caused by people or process failures, the most common cause of which is change. Industry analysts indicate that "unmanaged change is one of the leading causes of downtime or missed service-level agreements (SLAs)." [1] Thus, we submit that it is more appropriate to *seek the most opportune time to execute the change operations in a distributed infrastructure, based on the change's impact on the service-level objectives* (e.g. response time, availability, and recovery time). Such an impact-sensitive change-management strategy aims to respect the overall performance and dependability guarantees of the running services, yet allowing the system to incorporate changes of various kinds.

Figure 1 illustrates the main elements of the change planning problem. In typical IT infrastructures, there are multiple kinds of change operations, originating from various sources. Some changes are planned in advance – e.g., deploying new applications, upgrading obsolete software, increasing the system capacity –, and are derived from an *external* request for change (RFC). In other cases, changes are due to "firefighting" (mitigating the negative effects of unplanned situations) and are triggered by *internal* system-management events – e.g., 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. The change-operation planner must produce a timed schedule for executing the changes and must consider the impact on all the relevant quality of service requirements, as expressed by service-level objectives (SLOs), along with the objectives of the change operation. Each objective has a specific *business value* metric (e.g., the penalties associated with a missed change deadline or with degraded performance) for gauging the utility of meeting the objective based on the level of service parameters, called Key Performance Indicators (KPIs). The change schedule must maximize the aggregated business value, associated with all the enterprise objectives. 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 the 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 across heterogeneous, off-the-shelf components that sometimes span independent administrative domains. In these environments, the high-level service objectives translate into component-level objectives that can be managed by component-specific configuration managers. For example, a workload manager prioritizes and routes the service requests, monitoring the response-time objectives, and a dependability manager primes backup

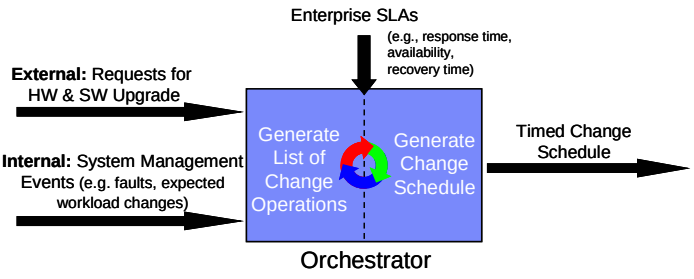

Figure 1. Dynamic change management problem.

nodes in anticipation of failures and performs recovery, monitoring the availability objectives. 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 [2].

As a result, we submit that the complexity and the distributed nature of objective management in real-world systems warrant a decentralized framework for dynamic change management. Building on this principle, this paper proposes a change management framework that separates the impact assessment from the change-operation scheduling. An *orchestrator*, responsible for building the schedule, consults multiple *objective advisors* (e.g., performance and dependability advisors) for assessing the schedule's impact on the service objectives. The advisors are software components that incorporate the domain knowledge to answer "what if" questions about service KPIs (such as performance and availability forecasts), given a description of the change operations and the timing of their execution. The orchestrator uses the returned predictions to compute the aggregated business value and to converge towards the optimal schedule through an iterative refinement process. The objective advisors themselves can be composite, third-party services.

The novel characteristics of this distributed framework for orchestrating change operations are:

- Accounting for both internal (e.g., faults, workload changes) and external (e.g., RFC) changes;
- Evaluating the long-term impact on performance and dependability objectives, both during and after the change execution;
- Optimizing the overall business value, which reflects the impact on the enterprise SLOs and on the change-request deadline.

Section 2 describes our distributed framework for dynamic change orchestration. Section 3 presents a case study of change management that we use to validate our architecture. Section 4 discusses the applicability of our approach for realistic systems and outlines directions for future work. Section 5 presents the relevant related work, and the conclusion summarizes our main ideas.

2. Dynamic change-management framework

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. In our framework, these tasks are performed by different components, which may come from different providers.

We assume that the objective advisors are able to predict future incoming loads, either because the workloads have a strong periodicity, or because fluctuations are preceded by recognizable patterns of warnings and notifications [3]. 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 services do not have hard real-time constraints (typical in enterprise systems).

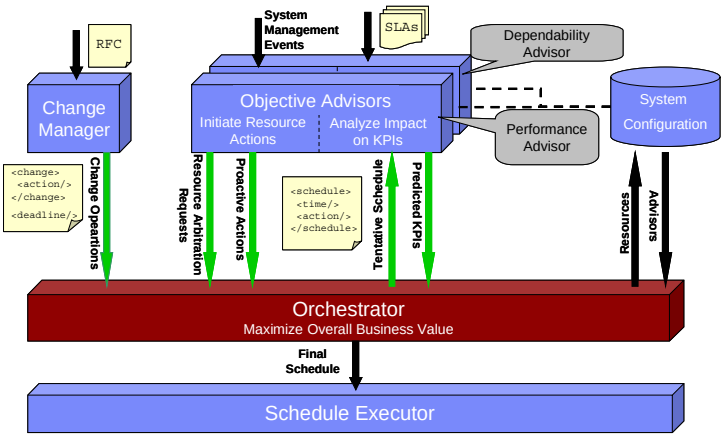

Figure 2. Distributed architecture for change management.

2.1. Framework components and protocols

Figure 2 illustrates the main components and interactions in our framework. The `ChangeManager` receives high-level RFCs, decomposes them into finer-grained change operations and related dependencies and forwards them to a centralized component, called *orchestrator*. The orchestrator receives the list of change operations and their execution constraints and generates a change plan through an iterative process. Distributed components called *objective advisors* analyze the impact of change plans; the orchestrator identifies the relevant advisors by querying the System Configuration Database. The objective advisors represent the service managers in the infrastructure and can use manager-specific knowledge to estimate the impact of a plan on the service KPIs. The orchestrator consumes these estimations and schedules the change operations with the objective of maximizing the overall business value. The interaction between the orchestrator and advisors is based on the Web Services standard, which guarantees compatibility in a complex system with components built by different providers. The orchestrator sends the final schedule to the `ScheduleExecutor`, which triggers the change operations at the indicated times. The `ChangeManager` is analogous to the Task Graph Builder from [7], and the `ScheduleExecutor` is similar to the TIO Provisioning Manager [6]. In this paper, we focus on the orchestrator and the objective advisors, which are novel to our approach.

Objective advisors. The objective advisors (e.g., performance and dependability advisors) exploit the functionality provided by the component-specific configuration managers. The advisors can be hierarchical and may span multiple administrative domains in order to manage end-to-end KPIs. The advisors estimate the impact

Table 1. APIs of the components.

Orchestrator	Objective Advisors
`InitiateChange()`: request for scheduling a group of change operations derived from an RFC.	`GetCurrentKPIs()`: request for current KPI predictions for a given time interval, assuming that only infrastructure events (e.g., workload variation, node failures) will occur.
`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).	`GetImpactKPIs()`: request for KPI predictions over a given time interval for a schedule of change operations.[a]
`ChangeSLA()`: request for integration of SLA updates.	

[a]The reply can suggest a set of *proactive actions* expected to improve the KPIs in conjunction with the change operations (e.g., "checkpoint database"). Proactive actions are included in the final schedule only if they improve the overall business value.

of observed, predicted or scheduled events on the service KPIs. They do not depend on the actual enterprise business value models, which are handled by the orchestrator.

Orchestrator. The orchestrator is a resource broker and a planner of change operations. The orchestrator is invoked in one of three situations (see Table 1): (i) when a change sequence has been initiated, following a RFC; (ii) when a predicted or observed infrastructure event (e.g., a fault, a workload change) mandates a resource reassignment; and (iii) when an SLA has changed, indicating a potential change in the overall business-value calculations. During the scheduling process, the orchestrator communicates with the objective advisors, asking "what if" questions 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 overall business value (which represents the utility of the schedule) using models defined by the SLAs. Using this metric to compare different schedules, the orchestrator converges, through an iterative process, to the best feasible schedule.

Interaction Protocol. The interaction protocol is at the heart of our approach. As shown in Figure 2, a change sequence is initiated by the ChangeManager with the `InitiateChange()` function, or by an advisor with `InitiateResourceBrokering()`. The orchestrator calls the `GetCurrentKPIs()` and `GetImpactKPIs()` functions of each of the advisors (see Table 1), creating and refining a schedule through an incremental process.

To minimize the communication costs, the interaction protocol might locally cache business value information for partial schedules. Each schedule receives a unique identifier, known to the orchestrator and advisors, and its related KPI predictions are saved. The predictions are retrieved whenever the partial schedule is modified, thus avoiding the repetition of most of the computations.

2.2. Business-value model

The SLO business values (BV) are functions that associate a dollar value with various levels of service provided by the system. They reflect the utility of a certain state of the system and provide a way of comparing the effects of changes affecting different KPIs. At time *t*, a KPI value is *KPI(t)* and the business value is: $BV_{SLO}(KPI(t))$. A KPI value is assumed to hold for a period of time, until some event causes the KPI to take another value. This means that *KPI(t)* is a step function. 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:

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

The business value functions of different SLOs are designed to be additive. They are used for reasoning about the multiple impacts of various change operations and for selecting the best trade-offs. We add the business values of all the SLOs to compute the overall business value, which reflects the utility of the proposed schedule of operations:

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

2.3. Change-operation scheduling

The orchestrator computes schedules for change-operation groups, which correspond to a request for change (RFC) or to a request for resource brokering. A schedule indicates when each individual operation from the group will start executing. The goal of the scheduler is to maximize, for a certain time-horizon, the overall business value.

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 imposed, etc. Change operations can be mandatory, such the operations derived from an RFC, or optional, such as the resource brokering operations. The scheduler can discard optional operations if they do not improve the business value.

Each change group defines a partial order between the operations, indicating their precedence dependencies. A group may also specify a deadline for completing the execution of all operations and a business-value expression reflecting the penalty of late completion, which will be factored into the overall business value of the system to be maximized by the orchestrator. If the deadline is missing, then the aggregated business value of the SLOs is the only criterion for selecting a schedule. A change-operation group can be preempted by the arrival of a group with higher priority (e.g., if a previous change has damaged the system and needs to be rolled back).

The orchestrator associates start times t_1, t_2 … t_n with operations e_1, e_2 … e_n, which have the respective durations d_1, d_2 … d_n. The schedule must comply with the partial ordering among operations and the group deadline D (if defined). During scheduling, the orchestrator queries the objective advisors for predictions of the impact on KPIs during the relevant time-horizon and uses these predictions to compute the overall business value and to refine the schedule. This process should generate a schedule providing the best possible business value.

Scheduling algorithms. Since the orchestrator does not know the closed form of the overall business value function (the KPI forecasts are done by the advisors), the scheduling problem has an unknown objective function. In the current stage of implementation, the scheduler makes a few simplifying assumptions about the change-operation groups: (i) all operations in a change group are mandatory; (ii) all the change-operation groups have explicit deadlines, like in the case of an external RFC; (iii) the operations in a change group are totally ordered (i.e. an operation must complete before the next one can begin). While these assumptions are somewhat constraining, we believe that in practice there are many change management situations that satisfy these constraints (we give an example in Section 3).

The algorithms we have implemented are based on the following pattern. For each operation e_k, we compute the feasible scheduling interval: $\sum_{i=1}^{k-1} d_i \le t_k \le D - \sum_{i=k}^{n} d_i$. Using these bounds, we try to schedule each operation at its earliest time, its latest time and at all the m prediction points (time instants indicating the future variation of the KPIs) that fall within this feasible interval.

The baseline scheduler is a backtracking algorithm that generates and evaluates all the possible placements for the change operations in a group. This algorithm generates the optimal schedule and has the worst-case complexity $O(m^n)$.

A more realistic scheduler uses a polynomial best-effort algorithm that is not guaranteed to provide an optimal solution. We achieve this with a greedy algorithm: we place each operation at each possible position and we compute the resulting business value (Figure 3). Then, we can select either the operation and the placement that yield the best possible business value (algorithm `Greedy1`), or the operation that displays the largest overall business-value variation depending on the scheduling time, in order to avoid giving priority to the short operations that have a small negative impact (algorithm `Greedy2`). This placement splits the timeline and the change-operation group in two, and the same algorithm is applied to the two segments of the problem. These two algorithms have the complexity $O(n^2m)$ because, for scheduling each of the n operations, they evaluate nm placement options.

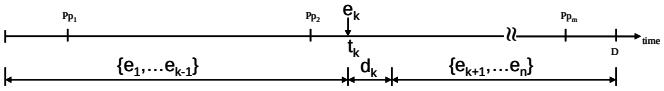

Figure 3. Greedy scheduling algorithm.

3. Case study

We consider a two-tiered system, where the physical hosts are organized in independently-managed node-groups. The first tier is a node group of application servers managed by application server middleware and the second tier is a node group of database servers, managed by database cluster infrastructure. The two node-group managers perform various middleware-specific management tasks (e.g., load balancing, request routing, fault recovery).

This infrastructure provides two services, each mapped onto corresponding application-server and database services. The two services processing Web transactions are load-balanced across three application-servers, W_1 to W_3. These front-end services query two database services that connect to separate database partitions. The database group comprises three nodes:

- 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 has response time, recovery time and availability objectives. The business value associated with these SLOs depends on the related KPIs, such as 'total number of transactions', 'number of transactions with response time below target', etc.

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 node-group managers (e.g., expected workload variations, service overheads). Similarly, a dependability advisor evaluates the impact on the recovery time and the availability SLOs.

3.1. Qualitative evaluation

For evaluating the proposed change management framework in this context, we have constructed a few realistic change management scenarios for this case study [3]. For lack of space, we present only one of them, involving a crash of node DB_1, and we discuss how our framework handles this scenario. We complement this analysis with measurements illustrating the trade-off between the cost and the loss of optimality of different scheduling algorithms (Section 3.2).

When the dependability advisor detects the crash of DB_1, the corresponding node-group manager takes immediate recovery measures. 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, and an accompanying higher risk of failing the availability objectives, the change-management system must decide whether removing one node from the

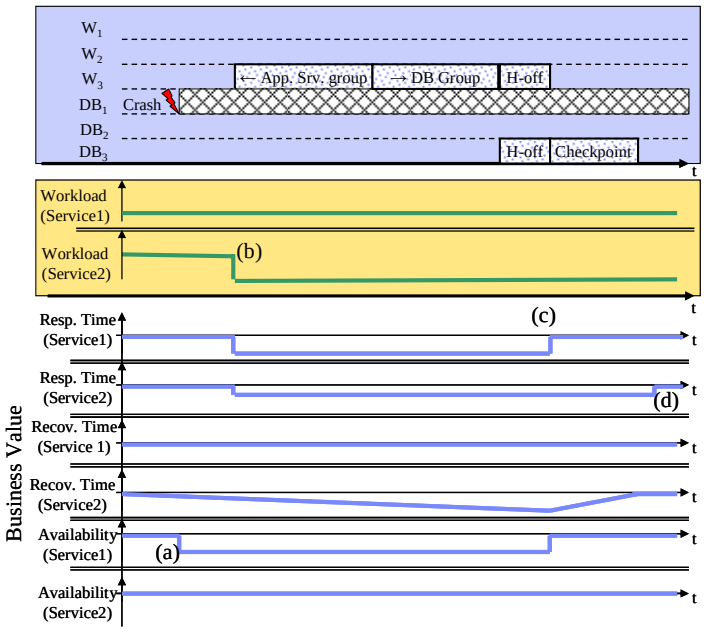

Figure 4. Node-fault management scenario.

application server group and adding it to the database group would improve the overall business value and when these operations should be scheduled.

Figure 4 shows the impact of these change operations. After the crash of DB_1, the lack of a backup leads to a sharp decrease of the predicted availability of Service1 and a drop in the corresponding business value – 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 fail to meet the response time objective. Therefore, the orchestrator delays the change operations 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. 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, the dependability advisor requests a proactive action in the form of a database checkpoint (synchronizing the modified data blocks in memory with the disk and shortening the log processed during recovery). After the checkpoint, indicated by point (d), the response time and the recovery time for Service2 decrease to normal operating levels.

Another scenario, involving an upgrade of the database, also shows that delaying the change operations may sometimes improve the overall business value [3]. These scenarios are typical of change management in an enterprise infrastructure; similar operations occur at a much larger scale in many real-life deployments. This illustrates the complexity of predicting the impact of change due to the strong dependencies on the actual implementations of objective managers. Our framework addresses these issues by delegating the impact assessment to objective-specific advisors encapsulating all the relevant domain knowledge.

3.2. Quantitative evaluation

Using a traditional scheduler, which does not optimize for long-term impact [6][7], in the scenario presented above 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 results in a missed opportunity for optimizing the overall business value. Instead, the scheduling algorithms presented in Section 2.3 find the optimal schedule for the first scenario. In this case, the run-times of all the algorithms –including the exponential backtracking scheduler – are comparable (less than 1s).

We also test our scheduler using randomly-generated input sets and we explore the trade-off between complexity and the loss of optimality. The most appropriate complexity measure is the number of times the business value needs to be evaluated, since these evaluations require communication between the orchestrator and the advisors; we do not report the run-times because they depend heavily on the hardware resources used for simulation. The loss of optimality shows how close the BV of the resulting schedule was to the BV of the optimal schedule, as generated by the backtracking algorithm. Figure 5 shows that, for small problems (e.g., 5 change operations and 10 KPI prediction points), the two (polynomial) greedy algorithms obtain near-optimal results and they need one or two orders of magnitude fewer BV evaluations than the exponential, optimal backtracking algorithm.

For larger problems we cannot use the backtracking algorithm and therefore we cannot measure the loss of optimality of the greedy 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` performs better than `Greedy1`, the two algorithms produced identical schedules for all but one of the randomly generated scenarios.

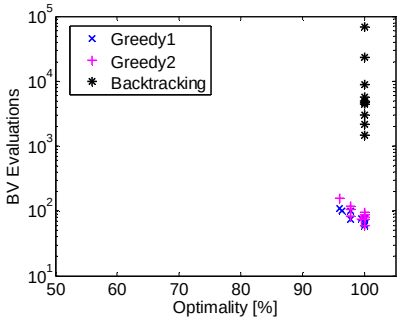

Figure 5. Cost vs. loss of optimality trade-off.

4. Discussion

By focusing on the communication protocol for impact assessment rather than on building a monolithic change management system, the proposed distributed infrastructure for change management facilitates changes that may span multiple independent administrative domains and that may target uncooperative software infrastructures. The generic orchestrator can communicate with third-party advisors, built with specific, proprietary domain knowledge about a service/system/vendor, and construct schedules using only the information 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 estimations 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 inevitably improve with accurate impact analysis. If the advisors provide incorrect information, the orchestrator might take the system to a state with unacceptable service levels; in this case, a downgrade or the rollback of the changes can be scheduled using the same process described above. Our approach mirrors the philosophy of Service-Oriented Architectures, which is to focus on interaction protocols, rather than implementation bindings.

One open question is to determine the typical size of realistic change-operation groups, which is important for selecting a good scheduling algorithm. Another issue to explore is the best way to express the KPI variation in time. The step function representation used in this paper might be too constraining; e.g., it cannot describe a recovery time that increases linearly with the increase of the database log, as depicted in Figure 4. We also plan to study, in a realistic setup, the impact of faults and upgrades, as well as the type of predictions and impact analysis that can be perform for impact-sensitive change management planning.

5. Related work

Segal and Frieder's seminal work [8] states that one of the general requirements for any dynamic updating system is supporting distributed programs that communicate across mutually distrustful administrative domains. Research, however, has mostly focused on mechanisms for implementing change at different levels of granularity (e.g., replacing components, objects, procedures), rather than on coordination of distributed changes and impact assessment.

Kharchenko et al. [5] estimate the "confidence in correctness" of composite Web Services undergoing online upgrades by monitoring multiple versions of a service in parallel. Some existing change management products, such as the IBM TIO [6], perform resource arbitration between node groups by evaluating the immediate impact of resource changes. While allowing the orchestration of distributed, self-managing components [2], this approach ignores the long-term impact of change management (e.g. interaction with expected workload change). CHAMPS [7] focuses on scheduling operations to satisfy external RFC deadlines. It develops a complex dependency-tracking framework and it formulates the scheduling problem as the optimization of a generic cost function given a set of constraints (representing the immediate impact of the change), providing a *centralized* approach for both scheduling and impact analysis. Our work is based on the observation that centralized impact evaluation is not appropriate for complex enterprise environments.

6. Conclusions

This paper investigates the problem of performing dynamic change management while maximizing the aggregate business value across all SLOs of the enterprise. We propose a novel framework for the distributed implementation of change management that separates the impact assessment (performed by the goal advisors) and the scheduling and business value aggregation (performed by the orchestrator). 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 validate this framework using realistic scenarios which emphasize that impact assessment is essential for maximizing the business value, and our simulations compare the trade-off between the cost and the loss of optimality of three scheduling strategies.

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 input during the early stages of this research.

References

[1] BMC Software, "Enterprise leadership: aligning IT and business as the economy rebounds," http://www.bmc.com/BMC/Common/CDA/hou_Page_Detail/0,3464,9926222_33961815_33965114,00.html

[2] D. Chess et al., "Experience with collaborating managers: node group manager and provisioning manager," *Second International Conference on Autonomic Computing*, 2005, pp. 39-50.

[3] T. Dumitraş et al., "Dynamic Change Management for Minimal Impact on Dependability and Performance in Autonomic Service-Oriented Architectures," *Tech. Rep. CMU-CyLab-06-003*, 2006.

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

[5] V. Kharchenko et al., "On Dependability of Composite Web Services with Components Upgraded Online," *Workshop on Architecting Dependable Systems*, Florence, Italy, 2004

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

[7] A. Keller et al., "The CHAMPS system: change management with planning and scheduling", *Network Operations and Management Symposium*, 2004

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