Ecotopia: An Ecological Framework for Change Management in Distributed Systems

Tudor Dumitraş[1], Daniela Roşu[2], Asit Dan[2], and Priya Narasimhan[1]

[1] ECE Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2] IBM T.J. Watson Research Center, Hawthorne, NY 10532, USA
tudor@cmu.edu, drosu@us.ibm.com, asit@us.ibm.com, priya@cs.cmu.edu

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 and have specific criticality/value models. In this paper, we present Ecotopia, a framework for change management in complex service-oriented architectures (SOA) that is ecological in its intent: it schedules change operations with the goal of minimizing the service-delivery disruptions by accounting for their impact on the SOA environment. The change-planning functionality of Ecotopia is split between multiple objective-advisors and a system-level change-orchestrator component. The objective advisors assess the change-impact on service delivery by estimating the expected values of the Key Performance Indicators (KPIs), during and after change. The orchestrator uses the KPI estimations to assess the per-objective and overall business-value changes over a long time-horizon and to identify the scheduling plan that maximizes the overall business value. Ecotopia handles both external change requests, like software upgrades, and internal changes requests, like fault-recovery actions. We evaluate the Ecotopia framework using two realistic change-management scenarios in distributed enterprise systems.

Keywords: Dynamic Change Management, Service Orchestration, Fault-Tolerant Architecture, Performability, Autonomic Computing.

1 Introduction

Enterprises demand highly available online systems and satisfactory service levels (*e.g.*, average response time) in the face of change. The kinds of changes that can occur are diverse, and can include recovery actions in response to failures, or upgrades due to new versions of software that become available. 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), rather than looking for the best time to do so. The downtime (or the perceived lack of responsiveness/availability) due to change management can disrupt the performance expectations of services and have an adverse effect on business.

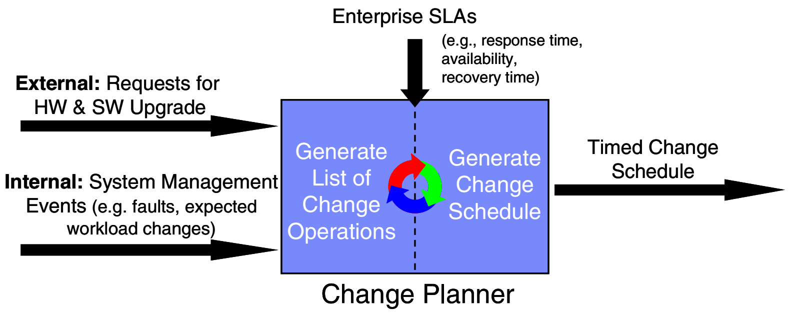

Enterprise SLAs
(e.g., response time, availability, recovery time)
External: Requests for HW & SW Upgrade
Internal: System Management Events (e.g. faults, expected workload changes)
Generate List of Change Operations
Generate Change Schedule
Timed Change Schedule
Change Planner

Fig. 1. Dynamic change management is likely to disrupt the critical services running in the IT infrastructure. Ecotopia handles changes based on both external requests (*e.g.*, software upgrades) and events detected internally by the autonomic management infrastructure (*e.g.*, faults) while taking into account their impact of the service-level agreements. The output is a timed schedule that seeks to wait for the most opportune time to apply each change operation and to maximize the enterprise business value.

Industry analysts indicate that "unmanaged change is one of the leading causes of downtime or missed service-level agreements (SLAs)." [1] Gartner Group states that “to address the 80 percent of unplanned downtime caused by "people failures," enterprises should invest in improving their change and problem management processes (to reduce downtime caused by application failures) and in automation tools, such as job scheduling and event management (to reduce downtime caused by operator errors).” [2] Thus, we hypothesize that it is more appropriate to *seek the most opportune time to execute the change operations in a distributed service-oriented 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.

Fig. 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” (*i.e.*, mitigating the negative effects of unplanned situations), and are triggered by *internal* system-management events, *e.g.*, faults or load 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 change-schedule for executing the changes and, in the process, must consider both the impact of the changes on all the relevant quality-of-service requirements as expressed by service-level objectives (SLOs), as well as the objectives of each change operation.

An SLO defines bounds and targets for a level-of-service metric (*e.g.*, response time, recovery time, availability), called Key Performance Indicator (KPI). An SLO also has a specific *business value* metric (*e.g.*, the penalties associated with a missed change deadline or with a degraded performance) for gauging the utility of fulfilling the objective [3]. The change schedule must maximize the aggregated business value, associated with all of the enterprise's SLOs. This optimization must span a long time-horizon, to account for both transient effects that might occur during the change execution, and permanent effects that might persist after the change has been finalized.

The change planner must be "ecological" in nature, *i.e.*, it must assess the impact of the change on the environment and its SLOs by considering a number of factors: the inter-dependencies among various system components, the available prior 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 by monitoring the response-time objectives, while a dependability manager primes backup nodes in anticipation of failures and performs recovery by 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 [4].

As a result, we believe that the complexity and the distributed nature of objective-management in real-world systems makes it unfeasible for a fully centralized change-operation planner to directly assess the impact of change operations on each service KPI. Rather, the impact on service KPIs should be estimated by the component-specific managers that control these services. However, component-specific managers might not be able to directly assess SLO business values necessary for estimating the overall change-impact, either because they do not directly implement the enterprise SLO models or because the service spans multiple managers and administrative domains.

Building on this principle, we propose Ecotopia, a change-management framework that decouples the impact assessment (handled by multiple objective advisors, *e.g.*, performance and dependability advisors) from the change-operation scheduling (handled by a change orchestrator). The *orchestrator* builds the change-operation schedule and estimates its business value impact based on the service KPIs predicted by the *objective advisors*. 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 properties associated with their execution. The orchestrator leverages the advisors' predictions to compute the per-objective and the aggregate business value, and to converge towards an optimal change-operation schedule through an iterative refinement process. The objective advisors themselves can be composite, third-party services.

The novel characteristics of the Ecotopia framework for orchestrating change-management operations are:

– Rich "what-if" interaction model that enables the use of fine-grained objective-advisor knowledge for an effective change scheduling decision. Our "what-if" model includes:
  - *Timeline of prediction points:* the advisors inform the orchestrator of the expected workload changes during the scheduling timeline. The orchestrator uses these guidelines to bootstrap the scheduling algorithms.
  - *Proactive actions:* the advisors can inform the orchestrator about specific actions that may improve the impact on KPIs during related change operations. The orchestrator can include these operations in the final schedule if they result in an improved overall business-value.

– Integrated management of both internal *(e.g.*, faults, workload changes) and external (*e.g.*, upgrades, capacity increases) changes. This approach is necessary because both types of changes affect a common pool of resources and services. Existing solutions [5, 6] assume different decision makers for the two types of changes.

– Complex business value functions for SLOs and change-request deadlines that can change along with the underlying enterprise service models, enabled by compliance with WS-Agreement standard [3]. Existing solutions support only priority-based models [5] or embedded, hard-coded utility functions [4, 7, 8].

– Optimization based on the long-term impact of change on performance and dependability objectives, accounting for both the time during and after execution of the change. Existing solutions consider only one of the two impact components (*e.g.*, [7] considers the impact during change execution, [5, 9] consider the impact after the change).

In Section 2 we compare Ecotopia with the state of the art in impact-aware change management. Section 3 describes the design of Ecotopia framework and Section 4 describes the current implementation. Section 5 presents two case studies of change management that we use to validate our architecture. Section 6 discusses the applicability of our ecological approach for realistic systems and outlines directions for future work.

2 Background

In their seminal paper, Segal and Frieder [10] 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. Their 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 [12, 13]. 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 the Ecotopia 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 effects of a fault). Additionally, the goal of our ecological framework is to manage the impact of change-management on the SOA environment (the running services and the existing resources). We assess this impact by asking and answering "what-if" questions about the outcome of the change operations. We assume some advance knowledge of the workload, as a running system has different behavioral profiles depending on the system load and the outcome of the changes will depend on the workload as well. Ecotopia tries to minimize the negative impact on the environment by using the answers to the "what-if" questions to determine the most opportune time to apply the changes, given the existing resources, the state of the running services and the workload.

2.1 Workload Prediction

Many workloads are characterized by a day-night periodicity [14]: the incoming request load increases during the day, with comparable peak request-rates from day to day, and decreases at nighttime to a very low baseline level. System administrators take advantage of this knowledge to over-provision the system for the highest expected loads [15] and to run maintenance activities (such as change management) during the night. There are also workloads with more complex patterns. The 1998 World Cup workload[1] [16] shows that the incoming load increases suddenly around game times, with lower peaks for the games played over a weekend. This trend is typical for sites dedicated to sporting events; this can be observed on Alexa.com[2] [17], by comparing statistics for two different sites covering the same event (*e.g.*, f1.com and fi-live.com): even if the peak loads are different, the access patterns are the same. On-line auction sites, such as ebay.com, exhibit similar load surges before the closing time of an auction. Furthermore, recognizable patterns of warnings and notifications that precede system events may facilitate the workload prediction [18, 19].

Ecotopia uses the ability to predict when the system is under high and low load for optimizing across multiple service-level objectives. For instance, an enterprise system

[1] This is the workload of a website dedicated the 1998 soccer World Cup in France. With 1.4 billion requests in the server logs, this is the largest web workload ever analyzed.

[2] Alexa is a tool for comparing statistics on the popularity and workloads of different websites.

may have two objectives: performance, expressed as average response time, and dependability, expressed as the expected recovery time after a system failure. After a fault (which, unlike a failure, does not completely disable the system), Ecotopia relies on knowledge of the workload to schedule the reconfiguration operations when the incoming load is low and avoid the penalties due to downtime during a busy period. Note that we do not assume that flash-crowd events (sudden load surges due to an unexpected increase in the site's popularity) are predictable; however, we show that exploiting irregular, but predictable workloads – such as the World Cup 98 trace [16] – allows Ecotopia to improve the scheduling of change operations when pursuing multiple objectives. Workload prediction is an optional part of the framework; Ecotopia's orchestrator can function with third-party advisors that answer "what-if" questions without providing workload predictions, *e.g.*, [8].

2.2 "What-if" Questions

Existing service-orchestration products [5, 20], perform resource arbitration between node groups by evaluating the impact just after the resource changes are enacted. While allowing the orchestration of distributed services [4], this approach is limited because it ignores the long-term impact of change management (*e.g.* interaction with expected workload change). The CHAMPS project [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 impact during change, *e.g.*, due to service unavailability), 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.

The problem of optimizing business value in a decentralized manner has also been addressed in the context of autonomic management of storage systems. Hippodrome [9] refines the initial configuration of a storage system through an iterative process, using a performance model to estimate the throughput and capacity of a particular configuration. Like our framework, Hippodrome separates between optimization and impact assessment, although the interactions between the two components are more tightly integrated and is based on a proprietary protocol. We submit that for complex systems integrating multi-vendor components we need an open communication protocol, for instance based on Web Services. The K2 middleware [21] goes further in distributing the autonomic management functionality by eliminating the centralized decision-maker and allowing individual "allocation pools" to manage their own objectives. In K2, distributed decision algorithms determine the goal configuration and the allocation pools start moving in that direction; if conditions change part-way through reconfiguration, the system changes its direction without having to invalidate the previous plan. However, none of these systems consider the evolution in time of the KPIs and the long-term impact of their decisions which are necessary for avoiding system instability and minimizing the overall business impact.

Thereska et al. [8] define a "resource advisor" predicting the impact of data placement and encoding choices on performance. The advisor has a hierarchical design, based on several "what-if" modules (for predicting the CPU, network and disk delays and cache hit rates) that can be combined together for end-to-end KPI predictions. Although it does not account for the detailed KPI evolution (it does not attempt to predict incoming request rates), the advisor continuously monitors the infrastructure and uses historical data to overprovision the system based on the peak loads observed. The authors report that prediction errors are less than 15% in most cases. This is an example of a third-party objective advisor that could be connected to the Ecotopia framework. Our orchestrator doesn't need to know the details of the performance models for storage systems; instead, it can use the "what-if" predictions to perform an ecological change management.

2.3 Timing the Application of Change Operations

The idea of waiting for the most opportune time to apply a change is widely accepted with respect to security patches for enterprise infrastructures. Beattie et al. [22] show that there is a sweet-spot for the time when security patches should be applied. Patches applied too early, without enough testing in the field, may introduce critical bugs or may conflict with local configurations. Patches applied too late leave the system exposed to security threats for an extended period of time. The authors argue that patching should be delayed until the risk of a security breach outweighs the risk of introducing bugs, and they develop a mathematical model for estimating the optimal time to apply a security patch.

Gorbenko et al. [23] 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.

Roşu et al. [24] introduce the approach of evaluating change plans based on actual SLO business values, which are computed by the orchestrator based on the service KPIs provided by objective advisors. Ecotopia extends this approach to a compressive "what-if" protocol appropriate for management of complex change requests. Other change orchestration solutions evaluate change plans in disconnection from the actual SLO of the enterprise, based on hard-coded utility models embedded in the resource advisors [4, 5, 8]. In [25], the change manager uses WS-Agreement specification to define business value parameters whereas the specification of the objective and business value functions is hard-coded in the orchestrator implementation. Neither of these approaches is appropriate for systems in which the objective and value models can evolve in time.

3 Design of an Ecological Change-Management Framework

A primary 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 a change-management system for enabling the control of change impact. The key to achieving this goal is the separation of scheduling and impact analysis. In Ecotopia, these tasks are performed by different components, which may come from different providers.

Service orchestration refers to an executable business process that combines multiple services by defining their interactions dynamically, with the goal of aligning the behavior of the composite service with the business objectives [20]. Ecotopia contains an orchestration engine that queries multiple objective advisors for KPI predictions and combines their outputs into a change-operation schedule. The predictions are based on detailed domain knowledge of each system component, but this knowledge is not exposed outside the objective advisors. Instead, the advisors answer simple "what-if" questions [8] about the impact of concrete change operations on service KPIs, considering the workload and the tentative schedules of these operations. The orchestration is driven by the enterprise SLAs, which define methods for computing the business value [3] that corresponds to the predicted KPI values. The business value reflects the utility of a given change schedule, allowing us to compare schedules and make an "ecological" choice: considering the impact on the IT environment, we select the change schedule that minimizes the service-delivery disruptions and that maximizes the overall business value.

General assumptions. We assume that KPI predictions can be derived from some knowledge of future incoming loads, either because the workloads exhibit clear trends [14, 16], or because fluctuations are preceded by recognizable patterns of warnings and notifications [18, 19]. Furthermore, we assume that the execution times of all the change operations submitted to the Ecotopia orchestrator can be estimated and that services do not have hard real-time constraints (which is typical of enterprise systems).

3.1 Framework Components

Fig. 2 illustrates the main components and interactions in the Ecotopia 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 the *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 planned change operations; the orchestrator identifies the relevant advisors by querying the `SystemConfigurationDatabase`. The objective advisors represent the service managers in the infrastructure and can use manager-specific knowledge to estimate the impact of a change plan on the service KPIs. The orchestrator consumes these estimations and schedules the change operations with the goal of maximizing the overall business value. The interaction between the orchestrator and the advisors is based on the Web Services standard, which facilitates 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

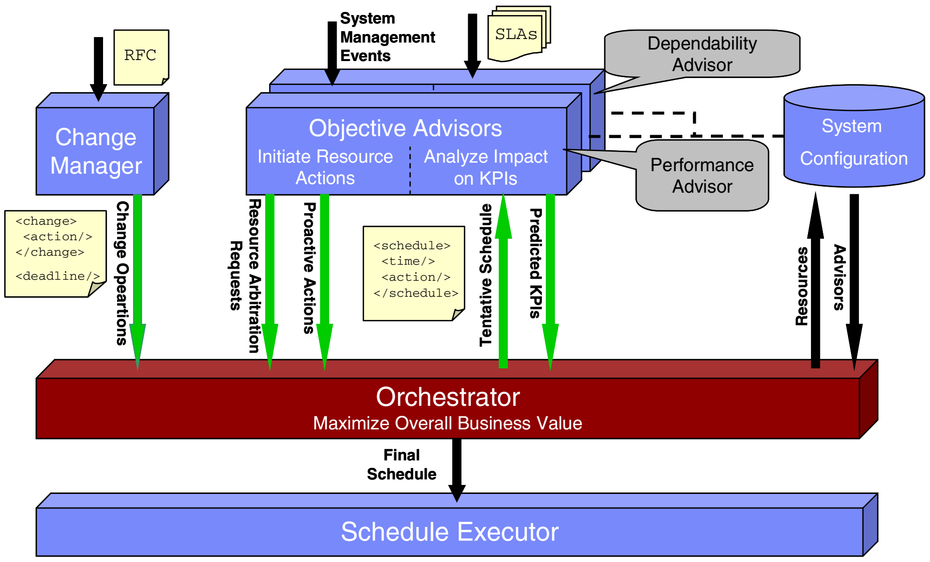


Fig. 2. Ecotopia's distributed ecological architecture for change management separates the tasks of impact assessment (performed by the objective advisors) and change scheduling (performed by the orchestrator). The orchestrator receives requests for change, queries the objective advisors with "what-if" questions about the tentative change schedule and uses the answers to refine the schedule with the goal of maximizing business value. The "what-if" interactions are based on an open protocol that allows the integration of third-party objective advisors.

from [7], and the `ScheduleExecutor` is similar to the TIO Provisioning Manager [5]. In this paper, we focus on the orchestrator, the objective advisors and their interactions, which are novel.

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 (in a similar manner to the resource advisor described in [8]). The Ecotopia advisors estimate the impact of observed, predicted, or scheduled events on a few service KPIs; for instance, we can define a performance advisor that predicts violations of the response-time objectives. The predictions do not depend on the actual enterprise business-value models, which are handled by the orchestrator.

The API of the advisors contains two functions, shown in see Table 1. `GetCurrentKPIs()` queries the KPI predictions if changes are not applied and it is used to assess the baseline for the change impact. `GetImpactKPIs()` retrieves the KPI predictions given a tentative change-operation schedule and is used to assess the impact the change schedule. These function invocations are synchronous (*i.e.*, the requestor waits to receive the KPI predictions before proceeding). The reply includes the KPI predictions for the entire time horizon of the decision. This might span multiple timeline points where the service KPIs change due to specific events such as expected workload changes or failures. These timeline points are called *prediction points*. The advisor reply includes one set of KPI predictions for each prediction point on the decision horizon. The replies can also suggest a set of *proactive actions* that are expected to improve the KPIs in conjunction with the change operations (*e.g.*, a "checkpoint database" action might reduce the recovery time). Proactive actions are included in the final change-operation schedule only if they improve the overall business value.

Orchestrator. The orchestrator is a resource broker and a change–operation planner. The orchestrator starts scheduling a group of change operations in two situations (see Table 1): (i) `InitiateChange()` indicates that a change sequence has been initiated, following a RFC; (ii) `InitiateResourceBrokering()` indicates that a predicted or observed infrastructure event (*e.g.*, a fault, a workload change) mandates a resource reassignment. All of these invocations on the orchestrator are asynchronous (*i.e.*, a response containing the schedule is not provided immediately). 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.

Table 1. APIs of the Ecotopia framework components

API	Description
Orchestrator	
`InitiateChange()`:	request for scheduling a group of change operations derived from an RFC.
`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).
`ChangeSLA()`:	request for integration of SLA updates.
Objective Advisors	
`GetCurrentKPIs()`:	request for current KPI predictions for a given time interval, assuming no change applied (*i.e.*, only infrastructure events such as workload variation or node failures will occur).
`GetImpactKPIs()`:	request for KPI predictions over a given time interval for a schedule of change operations.

Based on the predicted KPIs, the orchestrator creates a tentative change-operation schedule and computes its overall business value (BV). The SLA defines service-level objectives based on the monitored KPIs (*e.g.*, a target for the average response-time) and associates a business-value function to each SLO (*e.g.*, a penalty for each request that misses the target). The orchestrator computes the overall BV for a particular state of the system by adding the business values of all the services and SLOs defined in the service-level agreement. A change schedule will modify the overall BV by altering the state of the system and its monitored KPIs. When the orchestrator needs to choose among several alternative options for changing the system (*e.g.*, whether to include a proactive action in the schedule or not; all the possible times for scheduling a change operation), it uses the overall BV to select the best change-operation schedule. The overall BV reflects the utility of a change schedule and provides a way of comparing the effects of changes affecting multiple KPIs and SLOs.

The orchestrator is also invoked when an SLA has changed through `ChangeSLA()`, which indicates a modification in the overall business-value calculations. The orchestrator retrieves the new SLOs and the corresponding BV expressions and automatically updates its scheduling engine (more comprehensive mechanisms for managing SLAs updates are described in [24]). This is a reflexive hook allowing the orchestrator to update itself. In this case the change is applied immediately or at a specified time in the future, so it does not go through the scheduling process. New service-level agreements are typically defined in order to re-align the business and IT objectives of the enterprise; therefore, the effect of the new SLAs must be reflected as soon as they are available.

The goal of change-operation scheduling is to maximize the business value for a certain time horizon. The Ecotopia 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 change operation from the group will start executing. Using the overall business value, defined in the current SLAs, to compare different schedules, the orchestrator converges, through an iterative process, to the best feasible schedule.

3.2 "What-If" Interaction Protocol

The interaction protocol is at the heart of the Ecotopia framework. As shown in Fig. 2, a change sequence is initiated by the `ChangeManager` with the `InitiateChange()` function, or by an advisor with the `InitiateResource Brokering()` function. The orchestrator initiates the "what-if" interaction by calling the `GetCurrentKPIs()`functions of each of the advisors to learn about their prediction points during the decision time horizon and to establish a baseline state for assessing the impact of the proposed schedules. Then the orchestrator creates and refines schedules through an incremental process. It invokes the `GetImpactKPIs()` functions on each of the advisors to acquire the KPI predictions necessary for assessing the impact of each of the proposed partial and complete schedules.

The orchestrator and the objective advisors exchange all the information about the current change group and change-operation schedule needed to asses the impact on the KPIs and to improve the schedule. Table 2 summarizes these parameters.

A *change operation* is defined by a name, a scope and a set of properties. The name is an enterprise-specific descriptor (*e.g.*, "Upgrade database software to version 10.0") recognized by all of the related objective advisors and service managers. The scope identifies the resources (*e.g.*, "database node DB_1") involved by the operation.

Table 2. Scheduling parameters

$CG(n, e_{1...n}, d_{1...n}, R, D)$	Change-operation group
n	Number of change operations in the group
e_i	Change operation
e_i'	Optional change operation
d_i	Duration of change operations e_i
$R(e_i,e_j)$	True if e_i must be executed before e_j
D	Deadline of the change group
m	Number of prediction points
Pp_k	Prediction point
T_H	Time horizon for scheduling and impact assessment
t_i	Time instant when change operation e_i is scheduled to begin.

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 as 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. The set of operations in a group may expand during the scheduling process due to the proactive actions suggested by the objective advisors; in general, proactive actions can be considered optional.

Each change group defines a partial order among its constituent operations, indicating their precedence dependencies. A group may also specify a deadline for completing the execution of all its constituent 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 information 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 a higher priority (*e.g.*, if a previous change has damaged the system and needs to be rolled back).

The orchestrator uses the current KPI predictions as scheduling guidelines. 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) and changes that have already been scheduled. These prediction points indicate the time instants when the objective advisors expect the KPIs to change. After the scheduling of a change group is completed, the advisors add its impact on the infrastructure to the current KPI predictions.

To minimize the communication costs, the orchestrator might cache business-value information for partial schedules. Each unique schedule is tagged with an identifier (similar to a hash key), known to the orchestrator and advisors, and its related KPI predictions are saved. The orchestrator retrieves the predictions whenever it modifies the partial schedule by adding one or more change-operations, and thereby avoids repeating most of the computations.

4 Ecotopia Implementation

In this paper, we focus on the implementation of Ecotopia's orchestrator. The objective advisors rely on functionality provided by component-specific configuration managers [4, 5, 26, 27]. These managers encapsulate the extensive, and sometimes proprietary, domain knowledge (*e.g.*, workload characteristics, resource-utilization models), needed for assessing the impact of change operations on the service KPIs. For evaluating our framework, we have developed configurable emulators for the goal-advisors. We implement the orchestrator and the objective advisors as Web Services, which means that the orchestrator can interact with any third-party advisors that support the "what-if" interaction protocol described in Section 3.2.

4.1 Objective-Advisor Implementation

While the orchestrator is a centralized component, the objective advisors are distributed. Ecotopia uses an objective advisor for each SLO of each service defined the service-level agreement. For example, a performance advisor monitors the service to assess the response time, and a dependability advisor assesses the recovery time and the availability based on the amount of redundancy available in the current configuration. We implement the objective advisors in our framework in a hierarchical manner: as each service is composed of several other services, the advisor that corresponds to a top-level service queries several lower-level advisors corresponding to the component services. Every resource from the IT infrastructure is treated as a service: the network, the CPU, the disk, etc. have service-level objectives specifying the target for a set of KPIs, such as response time, throughput and recovery time.

The service composition and the mapping of services onto physical resources define a request queuing-path for each service. A change operation modifies this queuing path, either by altering its structure (*e.g.*, by defining a new service composition), or by modifying the parameters of the component queues (*e.g.*, by replacing a CPU with a faster one or by removing a replica from a load-balanced system). The advisors use this domain knowledge to answer "what-if" questions about service KPIs (such as performance and availability forecasts), based on the description of the change operations and the schedule.

The advisors corresponding to the primitive services contain analytical models of the corresponding resources and estimate the value of the KPIs based on the workload and configuration. For instance, the performance advisors estimate the response time of a primitive resource using the operational laws of queuing theory [28, 29], based on the incoming request rates and the known peak throughput of the resource. Higher-level advisors compute their KPI predictions by combining the outputs of the lower-level advisors along the corresponding queuing path. The composite queuing paths can be either sequential (*e.g.*, a request travels through a front-end, a local-area network and then a back-end) or parallel (*e.g.*, a load-balancer forwards the request to one of several servers for further processing). The parallel queuing paths do not necessarily have the same length; for instance, a request for a data item present in a proxy cache has a shorter path than a request that results in a cache miss and that

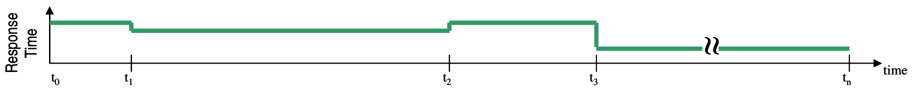


Fig. 3. A KPI (*e.g.*, average latency) varies in time, depending on the workload and the system configuration. We represent this variation by a vector of <*t, KPI(t)*> pairs indicating the time when a KPI changes and the new value. This corresponds to a step function as shown in the figure.

needs to be forwarded to the application server for processing. The parallel queues have probabilities associated with each alternative path representing the percentage of requests that travel along those paths.

Our implementation is similar to the resource advisor described in [8]; in addition, we leverage workload predictions to estimate the long-term KPI variation. KPIs change in time; therefore, the advisors provide KPI estimations as time-varying functions *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, as shown in Fig. 3. When replying to the invocation of `GetCurrentKPIs()`, the objective advisor will provide a list of pairs <Pp_k, $KPI(Pp_k)$>, indicating the times (prediction points) Pp_k when the KPI is expected to change and the corresponding KPI values (see Table 2). `GetImpactKPIs()` returns a similar list, indicating the effect of the suggested change schedule on the KPIs, computed using the service queuing-path created by the change.

4.2 Orchestrator Implementation

The orchestrator generates change-operation schedules, which associate start times t_1, t_2 ... t_n with operations e_1, e_2 ... e_n, respectively, which have the respective durations d_1, d_2 ... d_n (see Table 2). 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. The time horizon T_H must be long enough to include the deadline D, but in general will be longer, in order to account for the KPI impact after the change has been executed. The aim of the scheduling process is to provide the best possible business value.

The orchestrator does not know the closed-form equation 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 [30]. 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 (a regular objective function is non-decreasing in the completion times of the change operations); there are few theoretical results for scheduling problems with non-regular objective functions. We therefore focus on approximate scheduling algorithms that make the best effort to compute a solution close to the optimal schedule.

Business-value model. The SLO business values are functions that associate a dollar value with various levels of service provided by the system. A service-level objective defines a target for a particular KPI. A service may have multiple SLOs (some of these objectives may track a common KPI, *e.g.*, the target bounds for average latency and maximum latency), and each SLO has a business-value function. Since the KPIs change in time (see Fig. 3), the business values are also time-variable functions. At time *t*, a KPI value is *KPI(t)* and the corresponding business value is: $BV_{SLO}(KPI(t))$. 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:

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

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:

$$BV_{All}([t_0,t_n]) = \sum_{\text{All } SLO_k} BV_{SLO_k}([t_0,t_n]) \tag{2}$$

Scheduling assumptions. In this paper, we make a few simplifying assumptions about our scheduling problem. First, we assume that all the operations in a change group are mandatory (there are no proactive actions). Second, we assume that all the change-operation groups have explicit deadlines. When not defined explicitly, the deadline can be fixed to 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). While these assumptions are somewhat constraining, we believe that in practice there are many change-management situations that satisfy these constraints (we provide an example in Section 5).

Scheduling algorithms. The algorithms we have implemented are based on the following pattern. Each operation e_k has a *feasible scheduling interval*, defined by the earliest and latest times when e_k can be scheduled to allow enough time for the prior and subsequent operations:

$$\sum_{i=1}^{k-1} d_i \le t_k \le D - \sum_{i=k}^{n} d_i \tag{3}$$

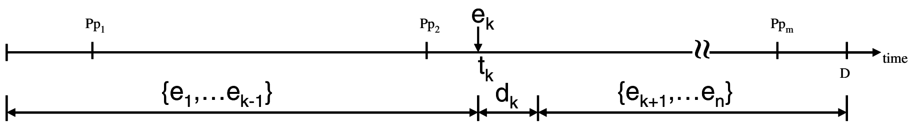


Fig. 4. Our Greedy algorithm for scheduling change operations first chooses the change operation e_k and the time t_k that yield the best business value. This placement splits the timeline and the change-operation group in two, and we apply the same algorithm to the two halves of the problem.

Using these bounds, we try to schedule each change operation at the earliest possible time, the latest possible 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 of the possible placements for the change operations in a group. We start with the first event e_1 and we place it at all the prediction points from its feasibility interval ($t_1=0$, $t_1=Pp_1$, $t_1=Pp_2$, etc.). For each of these values of t_1, we repeat the algorithm for the remaining operations and the new boundaries of the timeline (since we have started with the first operation, the deadline stays the same and the start time becomes t_1+d_1, the time when e_1 will complete). When we have successfully scheduled all the operations from the change group, we compute the corresponding business value by invoking `GetImpactKPIs()` on the relevant advisors. We then backtrack to try other possible placements of e_n, then of e_{n-1} etc., and we save the schedule that generates the highest business value.

If the KPIs are expressed as step functions, as shown in Fig. 3, 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. For each operation e_k, there may be *m* assignments of t_k. An assignment of t_{n-1} will be tested in combination with *m* assignments of t_n. An assignment of t_{n-2} will be tested with *m* assignments of t_{n-1}, each of which will be tested with *m* assignments of t_n; therefore, an assignment of t_{n-2} requires m^2 more operations for determining the best corresponding business value. By induction, this algorithm, henceforth called `Backtracking`, 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 e_k at each prediction point from its feasibility interval and we compute the business value that corresponds to this placement (during this step, we are only interested in the impact of e_k, so we invoke `GetImpactKPIs()` on the relevant advisors for a schedule that contains only e_k). We 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 recursively to the two segments of the problem, as shown in Fig. 4. Operations $e_1 \ldots e_{k-1}$ will be scheduled between $[0, t_k]$, and operations $e_{k+1} \ldots e_n$ will be scheduled between $[t_k+d_k, D]$.

The first iteration of this algorithm performs *nm* BV comparisons. In the worst case, the timeline partitioning will be skewed such that e_1 will be chosen and all the prediction points will fall after t_1+d_1. the second iteration will then require $m(n-1)$ BV comparisons. Since there are n iterations, this algorithm (`Greedy1`) has the complexity $O(n^2m)$.

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 in the following manner: at each iteration, we choose the operation e_k 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: $O(n^2m)$.

Schedule Stability. The schedules generated by the orchestrator remain constant in the absence of any additional change requests, SLA updates or system management events such as faults or workload changes. Fig. 5 shows that all the changes that might affect the final schedules are always initiated outside the scheduling loop involving the orchestrator and the advisors, which ensures the stability of our protocol. The advisors generate deterministic KPI predictions for a given change group (*i.e.*, the same tentative schedule will yield the same predictions).[3] The predictions returned by `GetCurrentKPIs()` will be adjusted in between change groups because the effects of the change that has just been scheduled are factored into the KPI predictions; however, no such adjustment is performed inside the scheduling loop. The algorithms presented above are guaranteed to converge if the KPI predictions are deterministic for a given change group. Other autonomic management systems based on iterative optimization loops [9, 21] may oscillate between borderline decisions because a resource reconfiguration will affect the performance metrics which may subsequently trigger another reconfiguration. Ecotopia, where all of the changes are initiated outside the scheduling loop and the "what-if" analysis considers a long time-horizon, guarantees that such infinite cyclic dependencies are broken and that thrashing cannot occur.

Canceling and Undoing Scheduled Changes. One corner case when the KPI predictions are not deterministic is when a fault or a load-surge prediction occurs while the scheduling loop is executing. Rather than updating the KPI predictions, in this case, we cancel the scheduling of the change group in order to avoid confusing the scheduler. Moreover, a fault or a load surge will typically be associated with a change request that has the highest urgency, so it is important to start scheduling this change as soon as possible. In general, whenever the orchestrator receives an urgent change request, it will preempt the currently executing scheduling process, and will start working on the new request immediately.

In some cases, it becomes obvious that a scheduled change does not have the desired effect and must be abandoned. If the change group has been scheduled but not yet implemented, it can be canceled easily. More often, however, this decision is taken only after the change has been finalized. In this case, another change has to be

[3] The interaction protocol described in Section 3.2 also relies on this property because the orchestrator and the advisors cache the KPI predictions corresponding to partial schedules.

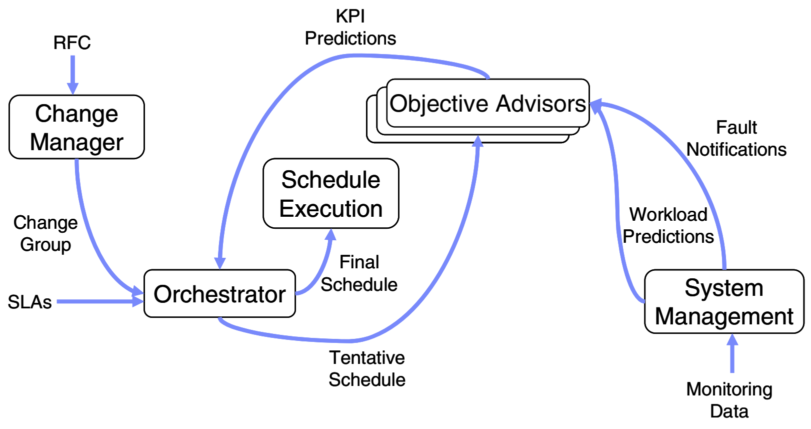


Fig. 5. The scheduling loop of Ecotopia is designed such that all the change requests originate from outside of the iterative interaction between the orchestrator and the objective advisors. This ensures that the scheduling process does not oscillate between borderline decisions.

scheduled to undo the effects of the previous one. The logs of the orchestrator can assist this operation by defining the reverse operations needed to undo the undesirable change, but the process must be guided by an administrator since the autonomic infrastructure has failed to take into account the negative effects of the change. In many cases, these errors are due to bad SLAs, which then have to be reworked by the system administrator. If the KPI predictions are accurate enough, we are confident that human interventions for correcting the orchestrator's decisions will be uncommon. Note that, since the decision to undo is not made by the orchestrator, the stability guarantees described above are not affected.

5 Case Study: Two-Tiered Enterprise Infrastructure

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 (*e.g.*, IBM WebSphere Extended Deployment [6]) and the second tier is a node group of database servers, managed by database cluster infrastructure (*e.g.*, Oracle Clusterware [27]). The two node-group managers perform various management tasks (*e.g.*, load balancing, request routing, fault recovery).

This infrastructure, illustrated in Fig. 6, 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, Srv_1 to Srv_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;

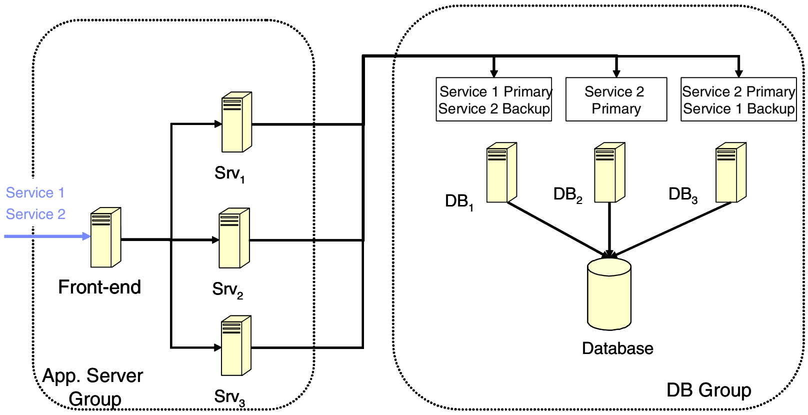


Fig. 6. Example: two-tier system

– 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.

5.1 Qualitative Evaluation

For evaluating the Ecotopia change-management framework in this context, we discuss two realistic change-management scenarios for this case study: a crash of node DB_1 and an upgrade of the database software. We complement this analysis with measurements illustrating the trade-off between the cost and the loss of optimality of different scheduling algorithms (Section 5.2).

Scenario 1: Hardware crash. 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 application server group and adding it to the database group would improve the overall business value and when these operations should be scheduled.

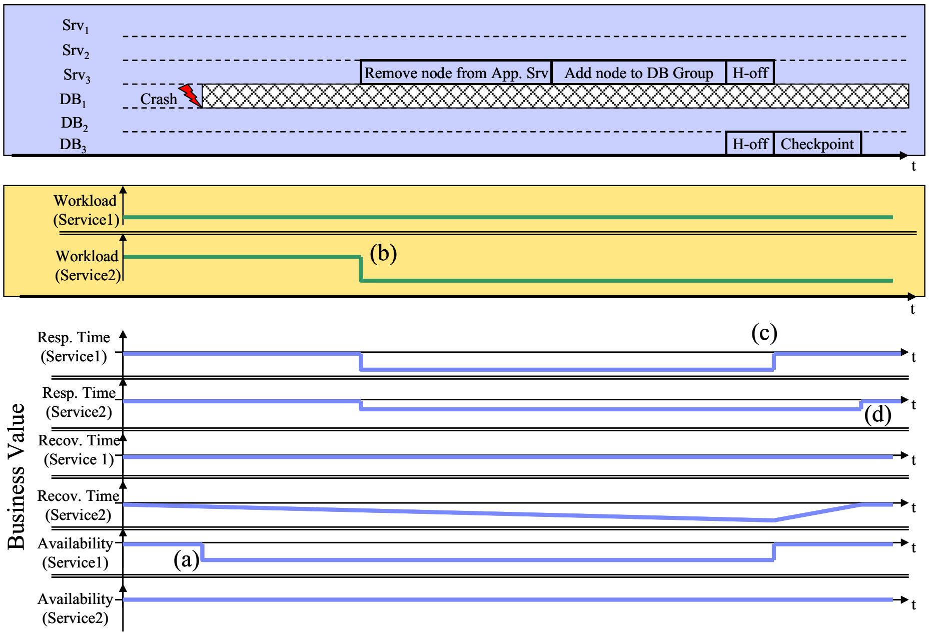


Fig. 7. Hardware crash and fault-management scenario

Fig. 7 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.

Scenario 2: Database upgrade. A similar impact analysis must be undertaken when upgrading the database software (Fig. 8). In this case, a request for change is decomposed into finer-grained 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

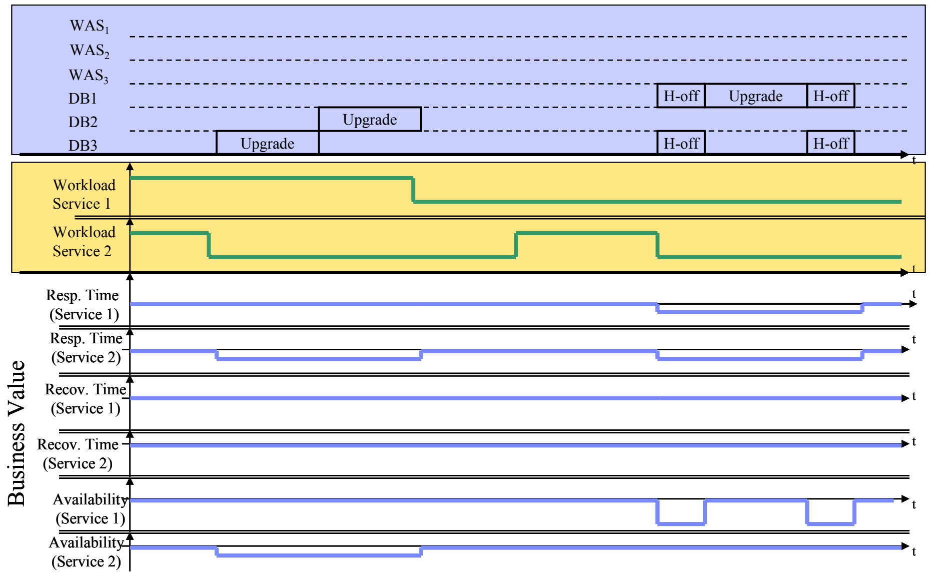


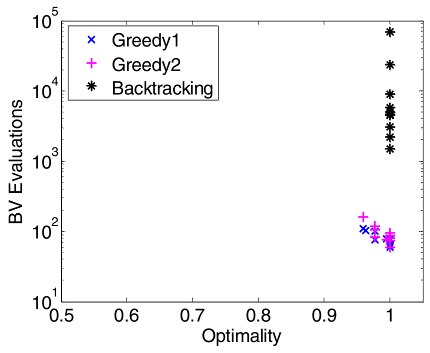
Fig. 8. Database-upgrade scenario

these operations on service objectives and their corresponding business values. 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 scenarios show that delaying the change operations may sometimes improve the overall business value. Such situations 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 that encapsulate all the relevant domain knowledge.

5.2 Quantitative Evaluation

Using a traditional scheduler, which does not optimize for long-term impact [5, 7, 9, 21], 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. The outcome of such impact-insensitive scheduling is a missed opportunity for optimizing the overall business value. Instead, the scheduling algorithms presented in Section 4.2



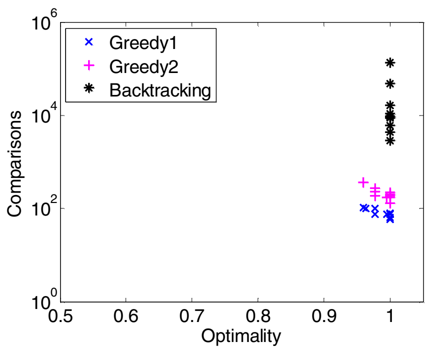


Fig. 9. Scheduling algorithms: trade-off between cost and loss of optimality. The Greedy algorithms are polynomial and yield schedules with a business value within 95% of the optimal achievable business value, which is computed using the exponential backtracking algorithm.

find the optimal schedule for these two scenarios and the run-times of all the algorithms – including the exponential backtracking scheduler – are comparable (less than 1s).

We also test our scheduler using several 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. Fig. 9 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.

6 Discussion

By focusing on the communication protocol for impact assessment rather than on building a monolithic change-management system, Ecotopia facilitates changes that might span multiple independent administrative domains and that might target heterogeneous software infrastructures. Our generic orchestrator can communicate with third-party advisors, which are built with specific, proprietary domain knowledge about a service/system/vendor, and construct schedules using only the information available from such advisors. This approach mirrors the philosophy of Service-Oriented Architectures, which is to focus on interaction protocols rather than on implementation bindings.

The separation between scheduling and impact assessment makes Ecotopia applicable to realistic systems, although it may limit its optimization capabilities when the advisors cannot provide a comprehensive impact analysis (*e.g.*, some services may not provide latency estimations, which are required for end-to-end response-time management). Moreover, the KPI predictions will inevitably have a degree of inaccuracy, especially when the time frame of the predictions is far ahead in the future. The orchestrator will generate change schedules even with imperfect information about the system; however, the quality of the schedules will 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. This raises two questions that we plan to investigate in the future: how much prediction inaccuracy can the orchestrator tolerate while keeping its ability to offer meaningful recommendations, and what kind of predictions and impact analysis can the advisors perform to enable ecological change-management planning.

Another open question is how to determine the typical size of realistic change-operation groups, which is important for selecting a good scheduling algorithm. The optimal scheduling-algorithm works well for the case study presented in this paper; however, we cannot use it for change groups with more than 10 operations, because of its exponential complexity. For very large problem sets, we may need to use heuristics such as genetic algorithms or simulated annealing [30]. We also plan to investigate the possibility of defining an adaptive scheduler that selects the best algorithm depending on the properties of the change-operation group (*e.g.*, its size).

The best way to express the KPI variation in time also warrants further exploration. 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 Fig. 7. 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 change points as scheduling guidelines, allows us to use simple algorithms even for a scheduling problem with an unknown objective function.

7 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 Ecotopia, a novel ecological framework for change management that tackles the complexity and the distributed nature of SLO management in real-world systems by separating the impact assessment (performed by the objective advisors) from the scheduling and business-value computation and aggregation (performed by the orchestrator). A novel "what-if" interaction protocol between advisors and orchestrator enables an efficient computation of SLO business values and change schedule refinement, Ecotopia performs ecological change management by taking into account the impact on the enterprise SLOs, the long-term KPI variations and the heterogeneous types and sources of change operations (both internal and external). We validate our framework using two realistic change scenarios that emphasize that impact assessment is essential for maximizing the business value. Our preliminary simulations compare the trade-offs 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, as well as Jean-Charles Fabre of LAAS CNRS for their input during the early stages of this research.

References

1. Kirkley, J.: Aligning IT and Business as the Economy Rebounds. Enterprise Leadership, BMC Software 2 (2004)
2. Gartner Group: High Availability Q&A: Failures, Standards and Metrics. Networked Systems Management Research Note QA-05-2701 (1998)
3. Global Grid Forum: Web services agreement specification (WS-Agreement). Draft, version 11 (2004)
4. Whalley, I., et al.: Experience with collaborating managers: node group manager and provisioning manager. Cluster Computing 9, 401–416 (2006)
5. IBM Tivoli Intelligent Orchestrator, http://www-306.ibm.com/software/tivoli/products/intell-orch
6. IBM WebSphere Extended Deployment, http://www-306.ibm.com/software/webservers/appserv/extend
7. Keller, A., et al.: The CHAMPS system: Change management with planning and scheduling. In: Network Operations and Management Symposium, pp. 395–408. Seoul, Korea (2004)
8. Thereska, E., et al.: Informed Data Distribution Selection in a Self-predicting Storage System. In: International Conference on Autonomic Computing, Dublin, Ireland (2006)
9. Anderson, E., et al.: Hippodrome: Running Circles Around Storage Administration. In: USENIX Conference on File and Storage Technologies (FAST '02), Monterey, CA, 13(2002)
10. Segal, M., Frieder, O.: On-the-fly program modification: Systems for dynamic updating. IEEE Software 10, 53–65 (1993)
11. Kramer, J., et al.: Towards Unifying Fault and Change Management. In: Workshop on Future Trends of Distributed Computing Systems in the 1990s, Cairo, Egypt, pp. 57–63 (1990)
12. Moser, L.E., et al.: Eternal: fault tolerance and live upgrades for distributed object systems. In: DARPA Information Survivability Conference and Exposition (DISCEX 00), Hilton Head, SC, pp. 184–196 (2000)
13. Bloom, T., Day, M.: Reconfiguration in Argus. In: Workshop on Configurable Distributed Systems, London, England, pp. 176–187 (1992)

14. Dilley, J.: Web server workload characterization. Technical Report HPL-96-160, Hewlett-Packard Laboratories (1996)
15. Vallamsetty, U., et al.: Characterization of E-Commerce Traffic. Electronic Commerce Research 3, 167–192 (2003)
16. Arlitt, M., Jin, T.: A workload characterization study of the 1998 World Cup Web site. IEEE Network 14, 30–37 (2000)
17. www.alexa.com
18. Pertet, S., Narasimhan, P.: Proactive Recovery in Distributed CORBA Applications. In: International Conference on Dependable Systems and Networks (DSN), Florence, Italy, pp. 357–366 (2004)
19. Zhang, Q., et al.: Workload-aware load balancing for clustered Web servers. IEEE Transactions on Parallel and Distributed Systems 16, 219–233 (2005)
20. Peltz, C.: Web services orchestration and choreography. IEEE Computer 36, 46–52 (2003)
21. Golding, R.A., Wong, T.M.: Walking toward moving goalposts: agile management for evolving systems. Hot topics in autonomic computing, HotAC, Dublin, Ireland (2006)
22. Beattie, S., et al.: Timing the Application of Security Patches for Optimal Uptime. In: Large Installation System Administration Conference, Philadelphia, PA, pp. 233–242 (2002)
23. Gorbenko, A., et al.: Dependable Composite Web Services with Components Upgraded Online. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems III. LNCS, vol. 3549, pp. 92–121. Springer, Heidelberg (2005)
24. Roşu, D., Dan, A.: Managing end-to-end lifecycle of global service policies. In: International Conference on Service Oriented Computing, Amsterdam, The Netherlands, pp. 570–575 (2005)
25. Keller, A.: Automating the Change Management Process with Electronic Contracts. In: International Workshop on Service Oriented Solutions for Cooperative Organizations, Yorktown Heights, NY, pp. 99–107 (2005)
26. WebSphere Extended Deployment Version 5.1 Information Center (2004)
27. Oracle Corporation: Oracle Real Application Cluster 10g. Oracle Technical White Paper (2005)
28. Lazowska, E., et al.: Quantitative System Performance: Computer System Analysis sing Queuing Network Models. Prentice-Hall, Englewood Cliffs (1984)
29. Urgaonkar, B., et al.: An analytical model for multi-tier internet services and its applications. In: International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Banff, Alberta, Canada, pp. 291–302 (2005)
30. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, Englewood Cliffs (2002)