A Dynamic Request Dispatching System for IT Service Management

David Loewenstern and Yixin Diao
IBM Thomas J. Watson Research Center
Hawthorne, NY 10532, USA
Email: {davidloe|diao}@us.ibm.com

***Abstract*— IT service delivery becomes an increasingly challenging business as customers demand improved quality of service while providers are driven to reduce the cost of delivery. While effective service delivery requires advances in many areas including workload management and workforce optimization, in this paper we focus on service request dispatching decision-making. Specifically, we propose an implemenation of a dynamic request dispatching system that continually adjusts the priority of service requests considering both (static) contractual service attainment targets and (dynamic) attainment levels achieved.**

I. Introduction

IT service delivery is characterized by strict service level agreements, an increasingly competitive environment, and ever narrowing profit margins. Consequently, service providers are continually searching for methods to improve quality of delivery while reducing cost of operations. These conflicting objectives lead providers to identify innovative methods for managing their business. Improved management of service attainment levels is one of the critical areas where providers seek improved performance, since missed service level targets result in both decreased customer satisfaction and significant monetary penalties. In this paper we propose a closed loop performance management solution for prioritizing workload with the objective of minimizing SLA violation and the associated penalties.

In a service delivery environment, each arriving service requests[1] has an associated contractual service level agreements (SLAs) that specifies the "quality of service" required for the incoming requests. Typically, service level agreements are structured such that SLA misses and penalties are not measured on a request-by-request basis, rather, against the performance of a group of arriving requests over a contractually specified time period. This leads to two distinctive management functions, namely, service operation management for daily request handling and service level management for periodic SLA performance calculation and reporting (typically monthly).

[1]Note that IT Information Library (ITIL) distinguishes between incidents (service interruptions such as database failure) and service requests (standard user requests such as password reset) [1]. However, in this paper we use service requests as a general term to refer to requests that have associated service level targets as documented in the Service Level Agreements. Similarly, we use service operation management to refer to management of daily service operation rather than Incident Management and Request Fulfilment as distinguished in ITIL.

However, separation of these two management functions results in reduced coordination across these two systems that impact one another's performance. In service operation management, incoming requests are prioritized and dispatched to service agents based upon the severity or target time of each request, irrespective of the attainment level achieved by the associated request class in the current evaluation period. In service level management, an SLA analyst team monitors and reports on SLA attainment levels at the end of each evaluation period which, in the case of SLA target miss, is too late to implement corrective actions.

In this paper we study a closed loop performance management scheme that integrates service level management and service operation management in order to implement a system that achieves proactive SLA management via dynamic dispatching priority adjustment.

Under the integrated system, dispatching decisions are made based on static information including request severity and SLA target time as well as dynamic SLA attainment levels reflecting the dynamic nature of workload variation and delivery capability changes. Our dynamic request dispatching scheme is in contrast to the static dispatching schemes such as priority (severity) based or earliest deadline first [2], [3]. We refer to these as static dispatching solutions since the dispatching decisions are based on static input (severity level or target time) but do not consider the dynamic system performance regarding the SLA attainment levels. While the static dispatching schemes are easier to implement, they suffer performance problems [4], [5]. For example, even if the requests from multiple customers have the same severity level and target time, they may have different SLA target attainment levels. As such, dispatching only based on severity level or target may overachieve for the customers with lower attainment levels but miss the SLAs for the customers with higher attainment levels. Following the algorithm studied in [5], in this paper, we focus on the implementation and integration challenges, and especially how we integrate this technology with other systems and elements in the service delivery environment and develop a complete performance management system.

The remainder of this paper is organized as follows. Section II provides a background on service delivery systems and summarizes the closed loop performance management algorithm. Section III presents the proposed dynamic request dispatching architecture. Section IV describes the implemen-

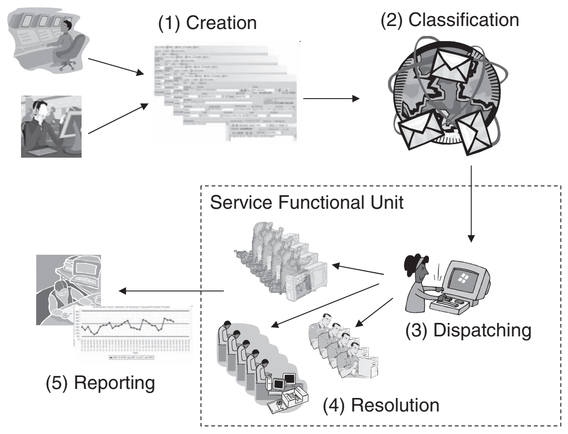

Fig. 1. Service delivery process.

tation drivers and implementation details. Our conclusions are contained in Section V.

II. Closed Loop Performance Management

A. Service Delivery Process

We begin by briefly describing the service delivery process as illustrated in Figure 1 which is comprised of the following five steps.

1) Creation: A service request arrives to the service delivery process by the end user (customer), the help desk personnel, the provider's monitoring team, or an automated alert monitoring system. Subsequently, a formal description of the service request (usually referred to as a "ticket") is created to document the request details.

2) Classification: To receive appropriate services, service requests are classified and routed to corresponding service functional units based on a fixed mapping that considers technology area and customer assignment.

3) Dispatching: Upon arrival to the service functional unit, service requests are reviewed by a dispatcher. The dispatcher prioritizes requests and assigns them to agents based on factors such as request severity, agent skill, and agent availability.

4) Resolution: The service agents respond to service requests to resolve and close the service requests.

5) Reporting: Request resolution time and backlog level are typically reported at the service functional unit level as health management and SLA attainment calculations are conducted by the customer account team as service level management.

B. Performance Management Algorithm

Successful service delivery management requires close integration between service operation management and service level management. As studied in [5], we propose a closed loop performance management scheme that integrates service level management and service operation management. Under the integrated system, dispatching decisions are made based on static information including request severity and SLA target time as well as dynamic SLA attainment levels reflecting the dynamic nature of workload variation and delivery capability changes. (This is referred to as a "closed loop" scenario where current SLA measurements influence future measurements via the dispatching decisions.) In this way, the service agents can be optimally utilized (to minimize SLA violation or the cost of SLA violation) and the SLA attainment targets from all request classes can be best met.

We use a feedback-based solution that dynamically adjusts request dispatching priorities based on the difference between the SLA attainment target and up-to-date attainment level measurement. The use of the closed loop system guided by feedback control principles provides a framework for a stable controller and can lead to robust performance against workload variation [4].

Specifically, we adopt an integral control framework to design a greedy algorithm for a "model-free" approach to determining the control gain. In addition to reducing the modeling complexity, the greedy algorithm is a reasonable approach because the decision space is limited and well defined. That is, there is only a small set of priority levels to explore. The greedy algorithm works as follows: (1) Set the initial request priority based on severity levels. For example, all severity 4 requests are set to priority 4. (2) In each control iteration, find the request class with the largest control error. Note that in the case where only one request class cannot meet its SLA target, $r_i - y_i(k)$ will be positive and the control error $e_i(k)$ will be positive as well. This request class will have the largest control error. In the case where all request classes can meet their SLA targets, the request class closest to its target will show the largest control error. (3) Increase the priority for the request class with the largest control error. (4)Dispatch service requests in the next control iteration based upon the updated priorities. Monitor service attainment levels and return to Step (2) for the next iteration.

III. Dynamic Request Dispatching Architecture

In this section we describe an architecture for dynamic dispatching decision making in a service delivery system based on the approach of Section II-B. The architecture was driven by two key requirements: minimizing the changes to the existing dispatching system, and supporting various usability options. Therefore, we chose the implemented architecture as a web service, separating from the existing dispatching system as well as being capable to be consumed both by human dispatchers and automation tools. As illustrated in Figure 2, the proposed dynamic request dispatching architecture integrates multiple existing and new systems including the ticketing systems, the dispatching system, and the SLA system.

The ticketing systems exist in the current dispatching process and are responsible for the intake or creation of service requests and their classification and routing to appropriate service units. One or more ticketing systems may be managed either by the customer or by the service provider. Once a service request has been created and assigned to a service unit, a subset of the service request information, including an abstract description, the request open date and time, customer and severity level, together with the service unit identifier, is stored within the Dispatching System as "ticket data." Normally, the process of copying this information from the ticketing system into the dispatching system is handled automatically through Simple Object Access Protocol (SOAP) remote procedure calls [6]. In other cases, however, the process is performed manually due to security concerns over providing a direct connection between the ticketing system and the dispatching system. As such, the ticket data need to be input through the presentation layer of the dispatching system.

The dispatching system also exists in the current dispatching process as the central hub for service request management. It interacts with the dispatchers to assign service requests to service agents and used by the service agents to update request status such as resolutions and closures. In some cases, the dispatching system helps to automate the dispatching process by maintaining service agent skills and availability schedules for the dispatcher as well. The dispatching system is contained in a web service environment providing web service interfaces (both SOAP and REST – Representational State Transfer [7]) as well as a presentation layer for human interactions. The dispatching system primarily consists of business objects that embody the business logic of entities used during dispatching. The examples of such entities include the service requests ("ticket data") and service agents ("SA data") with properties such as availability and skills. The business object layer is backed by a database, which maintains the current state of all entities represented by the business objects.

The SLA management system are newly introduced in the proposed integrated architecture to support closed loop performance management. It consist of three main components: the SLA attainment targets, the SLA attainment status, and the closed loop performance management (CPM) controller. The SLA attainment targets are entered into the SLA management system through the presentation layer by the SLA management team. This only needs to be performed occasionally when the new or updated targets are created. In contrast, the SLA attainment status needs to be read from the dispatching system (through REST/JSON) most often according to a pre-defined sampling frequency, typically once or multiple times a day, in order to get the real time performance measurement. With both SLA attainment targets and SLA attainment status as input, the CPM controller operates based on the control algorithm described in Section II-B in order to set service request priorities and minimize SLA attainment target failures. These priorities are sent back to the dispatching system in the form of a priority table. The priority table can be consumed either by the dispatching system to automatically set the service request priorities or by the dispatchers to manually set the request priorities.

Note that there is no direct connection between the SLA management system and the ticketing system because closed loop performance management acts as an internal process of the service provider; while the customer is involved in setting the SLA targets, this process is mediated by the service provider's SLA management team to fulfill the targets in an incremental and continual means.

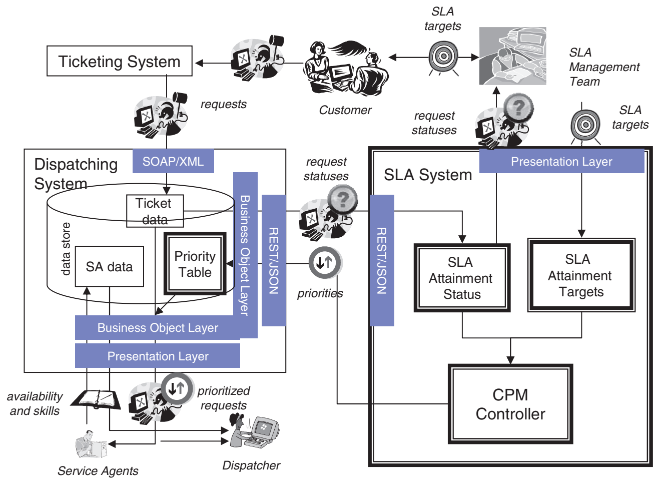

Fig. 2. Integrated system architecture for closed loop performance management. Single boxes represent existing components and systems, while doubled boxes represent new ones.

The process of determining the SLA attainment status from ticket data, using the attainment status together with the attainment targets to update the priority table, and using the priority table to schedule assignments to service agents forms the closed loop of service request performance management. From the perspective of the dispatching process, the addition of the closed loop performance management process adds an additional step when the dispatcher reviews the ticket data and assigns the corresponding requests to an appropriate and available service agents. By associating the ticket data with corresponding priorities from the priority table, the dispatchers influence the scheduling of service requests assignments with a longer term vision of how the customer's SLAs will be met at the end of the evaluation cycle. This is in contrast to the existing process where the service request is handled in an "isolated" way where the relative priority of the current service request is only compared with other opening requests.

IV. System Implementation

The architecture requirements described in Section III drove the implementation details to a large degree. The new SLA management system need to communicate with the existing dispatching system through pre-existed interfaces and also need to provide a browser window that is available to the dispatchers. Fortunately, the existing dispatching system supports web service interfaces, so that much of the implementation could be pulled together from standard components. In addition, the new CPM implementation need to extend the existing dispatching implementation with a minimum of changes to allow the piloting of CPM implementations in some service units without disrupting the non-CPM process in the remaining units. Finally, the new implementation has to encourage stakeholders (dispatchers, service agents, SLA management teams, and customers) to "buy in" to the CPM process. In practice, this means that CPM has to be usable as a decision support system even though it was intended to automate prioritization.

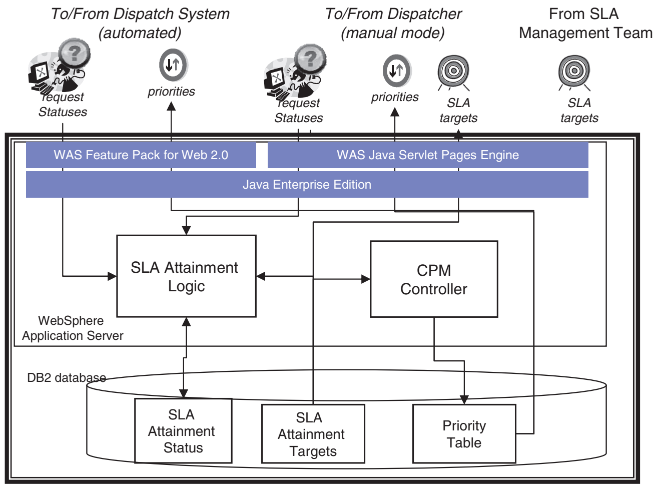

Fig. 3. Implementation of the SLA management system.

Automation of the dispatch process as a whole varies a great deal among customers and service units due to technical and security concerns, and the new implementation has to mesh cleanly with all levels of automation.

These drivers led us to a design that placed a premium on separation of closed loop performance management functionality from the rest of the dispatch process, and of communications and presentation from the closed loop performance management (CPM) algorithm. Figure 3 presents the resulting implementation in more detail. The CPM controller and SLA attainment logic are written in Java and hosted within a WebSphere Application Server (WAS) including Java Enterprise Edition (Java EE), and the attainment targets and status are stored as tables in a DB2 server. IBM Websphere Feature Pack for Web 2.0 [8] provides a JAX-RS library [9] for a RESTful JSON interface [10] to the dispatching system; WAS includes a Java Servlet Page (JSP) engine, which provides an HTML form interface and HTML/CSS displays for manual dispatching. The SLA attainment logic receives the service request status from both REST and JSP; the status are then compared with the attainment targets and provided as SLA attainment measures to the CPM controller. The CPM controller compares the SLA attainment measures to the SLA attainment targets to maintain a priority table for the service requests dynamically; this table is made available via REST, where it can be combined directly with ticket information in the dispatching system, and also via the presentation layer where it can be displayed to a dispatcher as a decision support tool. The SLA attainment targets themselves may also be displayed to help the dispatcher understand the calculated priorities and, if necessary, override them. The SLA attainment targets, the SLA attainment status, and the priority table are maintained directly through storage in DB2 database tables. Our data requirements placed a premium on consistency, with availability less important, and partition tolerance essentially irrelevant. Therefore, we rely on a relational database model, and specifically, DB2.

From a technical standpoint, the integration of the closed loop performance management system requires adding a business object class to represent the priority table. Where permitted, the priorities are then merged with the corresponding ticket data as “prioritized requests” through the business object layer when the tickets are accessed internally or displayed. The proposed architecture rules out changes to the dispatching system, including its implementation. Nonetheless, it is worth describing the dispatching system implementation briefly to understand the interaction between the SLA management system and the dispatching system. The dispatch system is implemented on top of Tivoli Service Request Manager (TSRM) [11] running on WAS with Java EE and DB2. TSRM includes SOAP and REST web services, JSON and XML parsers, its own presentation layer implementation, and a robust business object layer implementation. Any business object can easily be configured to expose any subset of its attributes through the web services. Integration of the SLA management system and the dispatching system user interfaces through the TSRM presentation layer is feasible but was not attempted since it would have reduced the ability to use the dispatching system without the SLA management system.

V. Conclusions and Future Work

In this paper we studied a dynamic request dispatching system for IT Service Management.

While the initial results are encouraging, there are several areas for further research. For example, several of the architecture requirements reflect the particular state of adoption of closed loop performance management and may be relaxed in the future. These requirements forced a tradeoff between efficiency in terms of computer resources and ease of integration with an existing implementation. In the future, moving CPM directly into the dispatching system could be of more benefit. In addition, current latency and throughput requirements for CPM are very modest, but could become much more stringent in the future. The CPM algorithm is naturally parallel at the service functional unit level, but deeper parallelization will require more research efforts. Similarly, the DB2 database implementation could be replaced with multiple geographically separated clusters. This would map better to the distribution of the service functional units and improve availability, and consistency costs should be modest since CPM operates at the service functional unit level.

References

[1] Office of Government Commerce, “IT Infrastructure Library. ITIL Service Support, version 3,” 2007.
[2] A. M. K. Cheng, *Real-Time Systems: Scheduling, Analysis, and Verification*. John Wiley & Sons, 2002.
[3] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.
[4] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley and Sons, 2004.
[5] Y. Diao and A. Heching, “Closed loop performance management for service delivery systems,” in *Proceedings of IFIP/IEEE Network Operations and Management Symposium*, 2012.
[6] T. Clements, “Overview of SOAP,” Oracle Sun Developer Network, 2002. [Online]. Available: http://java.sun.com/developer/technicalArticles/xml/webservices/

[7] A. Rodriguez, “RESTful Web services: The basics,” IBM developerWorks, 2008. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[8] “WebSphere Application Server Feature Pack for Web 2.0 and Mobile,” IBM. [Online]. Available: http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/web20-mobile/

[9] “JSR–000311 JAX-RS: The Java^{TM}API for RESTful Web Services (Final Release),” Java Community Process specification. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html

[10] D. Crockford, “JSON: JavaScript Object Notation,” Technical report, json.org, 2006.

[11] “Tivoli Service Request Manager,” IBM. [Online]. Available: http://www-01.ibm.com/software/tivoli/products/service-request-mgr/