A Framework for Real-Time Service-Oriented Architecture

Mark Panahi, Weiran Nie, and Kwei-Jay Lin
Department of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, CA 92697-2625, USA
{mpanahi, wnie, klin}@uci.edu

***Abstract*—Service-oriented architectures (SOA), though widely accepted in a variety of industries, must be enhanced to support real-time activities in order to gain even greater adoption. We present RT-Llama, a novel architecture for real-time SOA to support predictability in business processes. Based on a user-specified process and deadline, our architecture, containing global resource management and business process composition components, can reserve resources in advance for each service in the process to ensure it meets its end-to-end deadline. This is facilitated by also creating a real-time enterprise middleware that manages utilization of local resources by using efficient data structures and handles service requests via reserved CPU bandwidth. We demonstrate that RT-Llama's reservation components are both efficient and adaptable to dynamic real-time environments.**

***Keywords*-real-time; service-oriented architectures (SOA); real-time enterprise (RTE)**

I. Introduction

Service-oriented architecture (SOA) is the prevailing software paradigm for dynamically integrating loosely-coupled services into one cohesive business process (BP) using a standard-based software component framework [1], [2]. SOA-based systems may integrate both legacy and new services, created by either enterprises internally or external service providers.

However, current SOA solutions have not addressed the strict predictability demands that many enterprise applications require, from banking and finance to industrial automation and manufacturing. Such enterprises, many of whom already embrace SOA for a large part of their systems, would greatly benefit from a comprehensive SOA solution that can also encompass their real-time applications. In other words, as SOA gains prominence in many domains, the confluence of real-time and SOA systems is inevitable. We must prepare SOA for meeting the predictability requirements of real-time enterprise systems.

In this paper, we present the RT-Llama project (as an extension of Llama [3], [4]) which meets the real-time enterprise challenge by enabling SOA users to schedule an entire BP, thus eliminating the risk of missed deadlines due to the over-utilization of resources. RT-Llama differs from previous service-oriented architectures in that it allows end-to-end BP deadline guarantees through advance reservations of local resources.

In order to make this work, we 1) design global resource management and composition components that reserve resources in advance for each service in a BP to ensure it meets its end-to-end deadline; 2) implement a CPU bandwidth management system for each host essentially dividing a CPU into multiple temporally-isolated virtual CPUs, allowing different classes of service with various levels of predictability; and 3) develop a pre-screening mechanism to decrease the likelihood of unsuccessful distributed service reservations.

The rest of the paper is organized as follows. Sec. II reviews the challenges of bringing real-time to SOA. Sec. III presents the RT-Llama RT-SOA architecture. We present the performance study of the RT-Llama implementation in Sec. IV. Related work is compared in Sec. V.

II. Background

A. *Scope of Real Time Applications*

RT-SOA is a relatively new and challenging field of study. While some aspects of SOA make its transition to real-time simpler, still other aspects pose serious challenges. One real-world problem that can use an RT-SOA solution is *algorithmic trading*. Algorithmic trading is defined in Wikipedia as "a sequence of steps by which patterns in real-time market data can be recognized and responded to." The performance requirements of algorithmic trading demand not only fast transactions but also predictable ones [5]. Therefore, it is an application which should be implemented as RT-SOA. Examples of algorithmic trading strategies include: 1) trading several positions in coordination and 2) breaking larger trades into smaller sequential trades to minimize market impact.

As an example scenario, we assume that a customer has a choice for each trade among several brokerage firms. Each brokerage firm offers three levels of service:

1) *institutional*: for high volume traders where small delays can result in large losses, 2) *premium*: for moderate volume investors, and 3) *individual*: for low volume casual investors. Moreover, two operations are permitted. The `getQuote` operation requires a ticker symbol on an equity and returns its price per share. This operation is usually immediate and not reserved. The `trade` operation takes the quantity of shares and whether it is a buy or sell order. This type of operation has strict execution requirements and will usually be reserved in advance.

We make the following assumptions to guide us on our initial work. In the future, we plan to explore the effect of relaxing these assumptions.

- **One-shot BPs.** Due to the variable availability of hosts for executing services, we currently assume a "one-shot" model for BP composition and execution. Typically, BPs are composed and then reused multiple times. However, since RT-Llama's composition process takes into account service availability, the composed BP is intended to be used only once. If a user has the same requirements, the BP must be regenerated, possibly selecting different services for each BP.
- **Best-effort Reservations.** The RT-Llama architecture is based on the advance reservations of service executions to guarantee timeliness. Currently, we assume that reservation requests are made well enough in advance of the actual service execution such that there is plenty of time to send the request after the reservation. Therefore, reservation requests themselves are not subject to real-time requirements. In later work, we plan to explore "just-in-time" reservations.
- **Adoption of BPEL.** Some real-time systems determine admissibility of a task once it arrives at the host (i.e., admission control). For RT-SOA applications, this may not be acceptable. If tight predictability limits are required for a BP, it would take only one admission control rejection or cost overrun to severely disrupt the BP's execution. However, the advantage of SOA is that the entire BP is encoded in a higher level language, such as BPEL. The execution path is known before hand and can be reserved in advance. Therefore, we leverage this information in our RT-Llama framework and build advance reservation mechanisms so that the risk of service rejections and cost overruns is eliminated.

B. Real-time SOA Support

In order to promise end-to-end predictability for any real-time distributed system, every subcomponent or dependency must also provide predictability. Furthermore, an RT-SOA framework poses even more challenges than any stand-alone or distributed real-time system. We have identified the following required support for an RT-SOA and discuss the subset of these issues we wish to address in this work.

- **Operating System.** Any real-time middleware framework must be built atop an operating system that provides real-time scheduling and a fully preemptible kernel.
- **Communications Infrastructure.** The communications infrastructure must provide predictability – a requirement that the existing Internet infrastructure currently does not provide. However, approaches like Differentiated Services [6] and Integrated Services [7], despite their lack of wide acceptance, provide a starting point for QoS-based communications.
- **BP Composition Infrastructure.** The SOA composition infrastructure must be able to generate a BP that satisfies both a user's functional and timeliness requirements. It must be able to negotiate such timeliness requirements with distribution middleware to ensure predictable BP execution.
- **Distribution Middleware.** The main purpose of a real-time middleware platform (like a real-time enterprise service bus (ESB)) is to ensure the predictable execution of individual service requests. Therefore an RT-SOA middleware platform must provide support for advance reservations and avoid overloading or overbooking its host's resources.
- **Client Infrastructure.** Many existing SOA deployments use a business process execution language (BPEL) engine, which is a centralized mechanism to coordinate all remote service interactions within the process. An RT-SOA solution must address any unpredictability that a BPEL engine may produce. Alternatively, there are also distributed coordination mechanisms that route from service to service without the intervention of a centralized apparatus, which may be more amenable to achieving RT-SOA.

In our current work, we focus on introducing predictability into two of the main areas mentioned above: *BP composition infrastructure* and *distribution middleware*. We leverage Real-time Java [8] and the Solaris 10 OS to provide real-time scheduling capabilities to our middleware. We leave the issues of integrating real-time networking and predictable SOA client infrastructures to future work.

III. Architecture

A. System Model

In contrast to some existing distribution frameworks, SOA can support multiple classes of service that can be

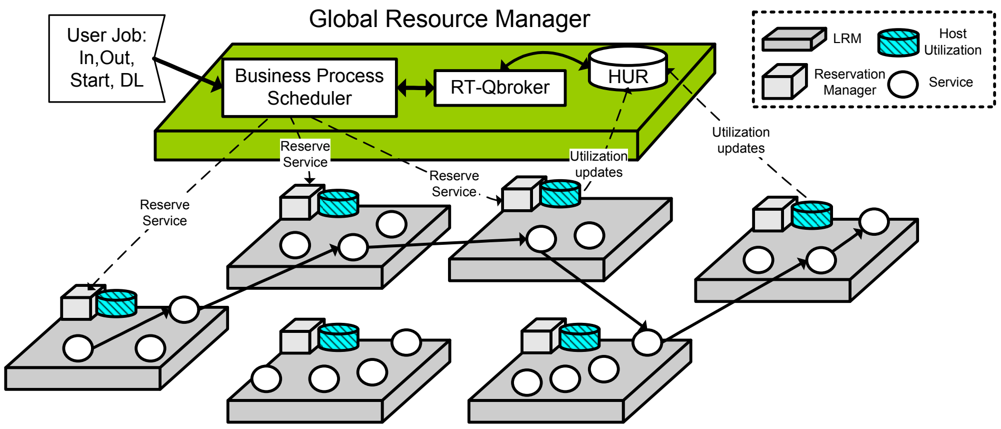

Figure 1. RT-SOA system using RT-Llama.

offered by providers at different price points. RT-Llama is comprised of several components that aid providers in offering predictable service execution and users in ensuring that their BPs meet their specified deadlines. A typical operating scenario is shown in Fig. 1. From a user's perspective, all that is required of them is to specify input values, desired output, and timeliness parameters, including start time and deadline. Based on this information, RT-Llama selects and reserves a feasible BP that matches the user's requirements.

In RT-Llama, providers can deploy services under two categories:

- **Unreserved:** Services deployed under this category accept requests without any prior workload reservation. There are two main subclasses:
  - **Immediate** requests are serviced in first-in first-out order according to the operating system's underlying real-time scheduler. There may be at most one immediate class. However, it may support several priorities.
  - **Background** requests are serviced in a best effort fashion according to the operating system's underlying non-real-time scheduler.
- **Reserved**: Services deployed under this category may only receive requests that have been reserved in advance. There may be any number of reserved classes. Reserved classes are intended to map to different levels of service that providers would like to offer. We identify three policies that may govern multiple classes:
  - **Resource favorability.** Providers can give higher classes more bandwidth (a larger share of system resources) than lower ones.
  - **Run-time spillover.** Providers can set policies such that higher class requests can steal bandwidth from lower classes in the case of cost overruns or tardy requests, in order to ensure that higher class requests meet their deadlines.
  - **Reservation spillover.** Higher class users may be able to have a service reservation span into lower classes allowing it to finish sooner and thus increasing the likelihood of a successful reservation of the overall BP.

B. Real Time Model

Similar to the standard real-time task model, we define the real-time task model for SOA as follows: A business process BP_i is a workflow composed of sequential service invocations $S_{i,j}$. Each BP_i begins execution at time r_i, finishes at time f_i, with an execution time of c_i, and has a deadline of d_i, that is respected if $f_i <= d_i$. Similarly, each service invocation arrives at time $r_{i,j}$, finishes at time $f_{i,j}$ after executing for $c_{i,j}$, with a deadline of $d_{i,j}$.

Since the RT-Llama framework supports advance reservations, we have a_i as the scheduled start time for BP_i and $a_{i,j}$ as the scheduled start time for $S_{i,j}$. We thus have the relation $w_i = d_i - a_i$ as the time window for BP_i and $w_{i,j} = d_{i,j} - a_{i,j}$ as the time window for $S_{i,j}$. Moreover, each $S_{i,j}$ has worst case execution time $wcet_{i,j}$.

This system model must be supported by the underlying infrastructure. Therefore, we currently assume that each host that deploys this framework is at least a 2-CPU or dual core system, with one (or more) CPU/core completely devoted to servicing real-time tasks (RT-CPU) and at least one available for background tasks and other operating system tasks (non-RT-CPU) as shown in Fig. 2. The RT-CPU will execute both the unreserved immediate class, as well as all reserved classes. These classes, although sharing the same CPU, must be *temporally* *isolated* from each other, meaning that cost overruns that may occur in one class cannot interfere with other classes. This is accomplished by creating multiple virtual CPUs out of the RT-CPU using the constant bandwidth server (CBS) framework [9]. Therefore, each class is assigned to a virtual CPU, which in turn has a system defined bandwidth percentage. For example, if we have one immediate class, and three reserved classes (H, M, and L), we may assign bandwidth allocations of 20%, 40%, 30%, and 10%, respectively.

C. RT-Llama Architecture

The RT-Llama architecture is shown in Fig. 2. Specifically, the components are as follows.

- The **Global Resource Manager (GRM)** components are responsible for scheduling a user's BP based on their requirements.
  - **QBroker/RT-QBroker** is a QoS broker designed to select a BP based on user specified constraints, and has been previously studied in [10]. In addition to its original role for best-effort BPs, RT-QBroker has been designed to perform feasibility checks on individual services during service selection, by consulting the Host Utilization Repository, to determine if the host is likely to be available during the general span of time of the BP.
  - The **Business Process Scheduler (BPS)** is responsible for reserving a BP selected by RT-QBroker according to either a concurrent or sequential mechanism (discussed later) by contacting the host of each service. In the event of a reservation failure (i.e., the reservation is not feasible or the request timeout is exceeded) the BPS requests a new BP from RT-QBroker, absent any unfeasible services. If RT-QBroker cannot find a feasible BP, the user's request is rejected.
  - The **Host Utilization Repository (HUR)** stores cached future utilization information regarding each host using TBTrees (discussed shortly). As a cache, it may be updated at predefined intervals, and thus not always completely up-to-date. It may also have coarser grain information than that stored on the host to save on space. However, it provides a quick way of determining if a service is likely to be available at a future time.
- The **Local Resource Manager (LRM)** is responsible for hosting services and ensuring that requests on such services are executed predictably by working with the GRM components.
  - The **Reservation Manager (RM)** manages advance reservations for the host. It uses a TBTree for each of the reserved classes to manage their overall utilization, as well as a simple hashmap to manage specific reservation information. Both data structures are represented by the **Reservation Repository (RR)**. The RM supports search, insert, and delete operations for reservations. Additionally, it sends asynchronous updates on recent utilization changes to the HUR.
  - The **Admission Controller (AC)** accepts incoming service requests and routes them to an **Executor** responsible for servicing requests for the various classes of service. Each executor is mapped to one or more threads on the underlying operating system. The real-time executors include the **Unreserved Executor** and the **Reserved Executors** and are each given a fixed amount of bandwidth on the RT-CPU by promoting the FIFO priority of the real-time thread backing the Executor for an amount of time out of a period according to its bandwidth percentage. For reserved requests, it looks up the $a_{i,j}$, $wcet_{i,j}$, and $d_{i,j}$ according to the request's reservation ID in the RR. This information is necessary for the Reserved Executors to ensure that requests are serviced in an earliest deadline first (EDF) fashion and to properly accommodate tardy requests and deadline overruns. The **Best-effort Executor**, by contrast, executes on the non-RT-CPU along with other system threads and provides no guarantees for predictability.

D. Reservation Data Management

One of the unique features of the RT-Llama framework in contrast to other SOA systems is its ability to reserve service executions in advance. This avoids the pessimistic nature of on-demand admission control strategies and leads to potentially higher utilization rates as users are able to plan their BP executions ahead-of-time.

To keep track of existing reservations in the RT-Llama framework, we favor the temporal bin tree (TBTree) discussed in [11] over other data structures due to its flexibility, efficiency in both time and space, and suitability for real-time planning applications. The TBTree uses a binary tree structure to store the total amount of time available within an interval at different levels of granularity. Each node in the tree contains the sum of the available time in its left and right children, each of which covering half the time interval of the parent. A search for available time begins at the root and descends into the tree and identifies

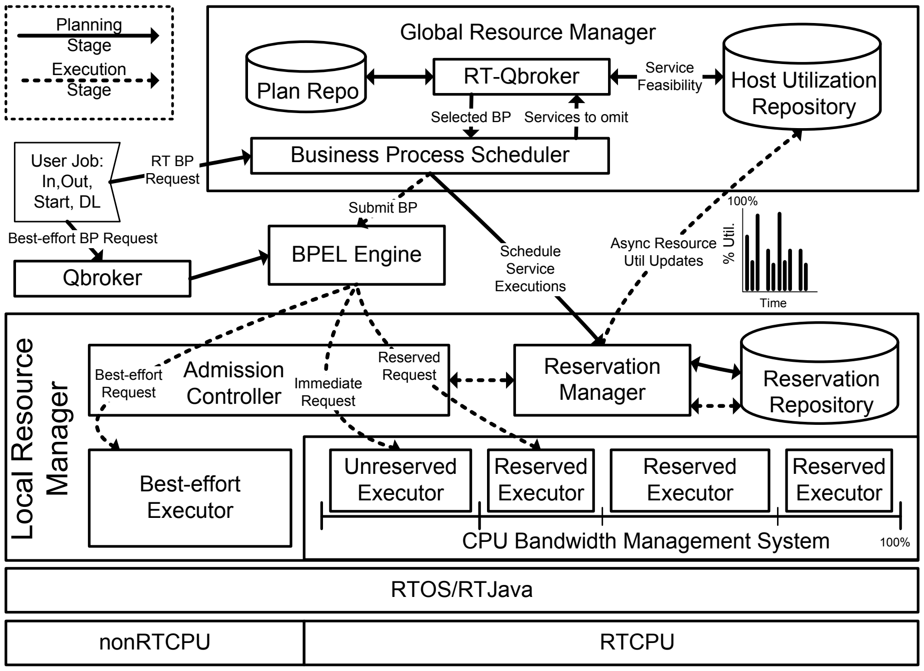

Figure 2. RT-Llama architecture.

the leaf node that can completely accommodate the task, specified as either an earliest start time or latest finish time, and the expected execution time.

E. Reservation Mechanism

As shown in Fig. 1, the reservation mechanism begins when a user submits the input, desired output, QoS requirements, start time and deadline to the GRM. The GRM in turn composes a suitable BP, schedules it for execution, and submits it to the BPEL engine at the specified start time. The main challenges during this process include 1) deciding the intermediate start times and deadlines for each service within the BP, and 2) appropriately reserving all the hosts involved in the BP using the two-phase commit transaction protocol. The exact difficulty with each challenge depends on the strategy being used.

When a BPS receives the initial input from a user, it then contacts RT-QBroker to construct the BP based on the functional requirements. During RT-QBroker's selection phase, it can pre-screen each service to be selected based on the service's utilization information in the HUR. The utilization of S_i's host over time t is given by $u_{i,j}(t)$, and let k be some utilization threshold. The service may be considered if its host's utilization over the BP's time frame is below the threshold, i.e., $u_{i,j}(w_i) \leq k$.

Once service selection is complete and a BP is composed, there are two main strategies that can be used to perform service reservations:

- **Concurrent:** This strategy is based on first selecting the BP and then determining the start time and deadline for each service, i.e., $a_{i,j}$ and $d_{i,j}$, after which the reservation process may begin. Moreover, both RT-QBroker and the BPS can screen service availability against the HUR. This way, if a service is unlikely to be available, RT-QBroker may produce another BP without the overhead of unsuccessful distributed reservation procedure.
  Before the reservation, the BPS must determine parameters $a_{i,j}$ and $d_{i,j}$ for each service. Under the concurrent strategy, we adopt two intermediate deadline assignment methods, *proportional deadline* (PD) assignment and *normalized proportional deadline* (NPD) assignment. For PD, the values of $a_{i,j}$, $d_{i,j}$, and $w_{i,j}$ are based on $wcet_{i,j}$ as a proportion of the total time available for the BP (i.e., w_i). Thus, we have $w_{i,j} = \frac{wcet_{i,j}}{\sum wcet_{i,j}} w_i$, $a_{i,j} = d_{i,j-1}$, and $d_{i,j} = a_{i,j} + w_{i,j}$. NPD is a more reasonable intermediate deadline assignment method in that it takes into account host utilizations. Under this scheme, we have $w_{i,j} = \frac{wcet_{i,j} * u_{i,j}(w_i)}{\sum_j wcet_{i,j} * u_{i,j}(w_i)}$. Once the intermediate start times and deadlines are computed, the BPS may apply a more precise screening process according to the relation $u_{i,j}(w_{i,j}) <= k$, essentially checking

the service's host's utilization over the service's own time window.

Now that the BP is ready to be reserved, asynchronous reservation requests are sent to the RM on each services' LRM, with a timeout attached to each request, using a transaction-based two-phase commit algorithm. If a timeout expires, or a reservation request is rejected, all pending service reservations must be undone, and the procedure repeats again with a new BP from RT-QBroker.

- **Sequential:** According to this strategy, a BP is reserved service-by-service from start to end using a greedy strategy in order to maximize the remaining time available for remaining services in the BP. RT-QBroker, and its screening option, is used in much the same way as in the concurrent strategy. However, the BPS does not find the intermediate start times and deadlines for each service. Timeout values are attached to each reservation request, and once a timeout expiration or reservation failure occurs, the BP reservation must be rolled back.

There are tradeoffs to both approaches. With the sequential approach, there will likely be a higher level of success with each attempt. Each attempt, however, will most likely take longer than the concurrent approach. But each attempt of the concurrent approach would be less likely to be successful as it relies at best on the cached information in the HUR.

IV. Simulation Results

The main goal of our simulation study is to observe the performance and tradeoffs of different reservation methods. To focus on the real-time requirements of BPs rather than the complexity of BP structures, all BPs in our simulation have only sequential structures.

We have defined the following system properties in our study.

1) *workload factor*: the amount of workload generated for the simulation. A workload factor of 1.0 represents a workload exactly equal to the system capacity, which is the maximum amount of work that can be handled by the system. A workload factor of 0.8 represents a workload 80 percent of the system capacity.
2) *success ratio*: the number of successful reservation requests divided by the total number of reservation requests.
3) *average attempts*: In our design, the GRM has the option of rejecting a request after the first unsuccessful reservation attempt or trying to come up with another BP and repeat the reservation. In other words, there could be multiple reservation attempts by the GRM per user request. We define the average attempts to be the total number of attempts divided by total number of requests, which approximates the average response time to a user request.

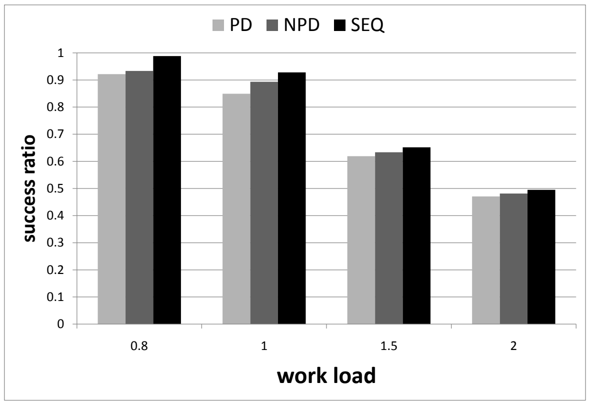

Figure 3. *Success ratio* for different reservation schemes and workloads (MaxAttempts=1, monotonic-random pattern).

In the following experiments, we simulate 5 hosts. A TBTree with 1024 time units is associated with each host to record the utilization information. Services with the same function are replicated so that we can always switch to a host with a low utilization if there is one. In the simulation, we simulate a process containing 3 services with each service having an execution time of 1 time unit. Process instances differ in their start times and have a relative end-to-end deadline varying between 20 to 32 time units. To simulate a more realistic environment, we pre-loaded the system with 30% of system capacity before collecting data.

As discussed in Section III-E, advance reservations can be made using: (1) concurrent with proportional deadline (PD) assignment, (2) concurrent with normalized proportional deadline (NPD) assignment, or (3) sequential (SEQ) schemes. We have designed experiments to see 1) the effect of the reservation methods and the *workload factor* on the success ratio, and 2) how many attempts on average are necessary to either accept or reject a reservation if we allow multiple attempts.

Fig. 3 shows the success ratio of the three reservation methods for different workload factors under a monotonic-random pattern. In a monotonic-random pattern, reservation requests form clusters that increase monotonically along the time horizon. But within each cluster, the reservation requests are random in their start times. We believe this work pattern simulates realistic workloads.

In Fig. 3, we can see that when the workload factor increases, the success ratio decreases because more requests are generated but the system capacity remains the

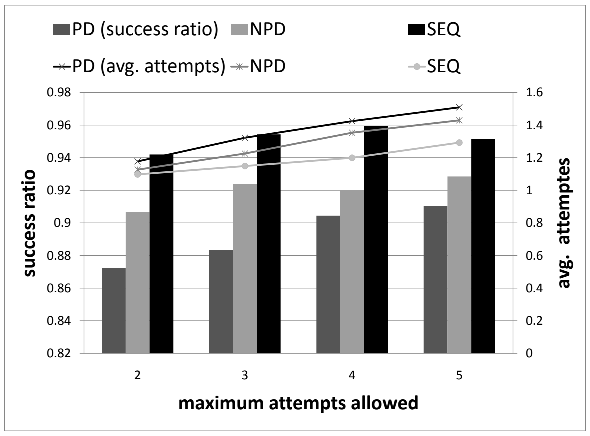

Figure 4. Effects on *success ratio* and *avg. attempts* using multiple reservation attempts (workload factor=1.0, monotonic-random pattern).

same. For example, the maximum success ratio that can be achieved for a workload factor of 2.0 is 50%. Another observation is that when the workload factor increases, the success ratios achieved by all schemes approach the upperbound value, i.e., 50% for a workload factor of 2.0.

Fig. 4 shows the effect on *success ratio* and *avg. attempts* if multiple reservation attempts can be made for a single business process request. As expected, *avg. attempts* (lines) increase for all three schemes when the max attempt value increases and the result for SEQ remains the lowest. For the *success ratio* (bars), concurrent schemes (PD and NPD) enjoy a relatively greater benefit from increasing the max attempts value, while SEQ remains high. Although SEQ outperforms both PD and NPD in terms of *success ratio* and *avg. attempts*, there is a tradeoff not shown in the figure: the response time for a sequential reservation is the sum of response times of individual reservations, whereas for concurrent schemes (i.e. PD and NPD), the response time is the maximum of response times of all individual service reservations. Therefore, the concurrent schemes may perform more efficiently where long communication latencies (between GRM and LRM) are expected.

V. Related Work

Real-time enterprise is an attractive idea that has received huge interest from many IT companies. Companies such as IBM, Microsoft, Sun Microsystems, and HP have all invested heavily on developing the technology. Microsoft has developed Microsoft Dynamics as a line of integrated business management solutions that automate and streamline financial, customer relationship, and supply chain processes. IBM has proposed the complex event processing (CEP) framework which is made up of WebSphere Business Events (the event processing engine), WebSphere Business Monitor (the rich dashboard), WebSphere Message Broker (event transformations and connectivity functions), and Generalized Publish and Subscribe Services (GPASS) [12]. The HP ZLE framework claims to provide application and data integration to create a seamless, enterprise-wide solution for real-time information and action. To our knowledge, however, none of the current real-time enterprise products offers service reservation capability.

The common object request broker architecture (CORBA) is an object-oriented distributed architecture designed to provide location, platform, and programming language transparency. RT-CORBA [13] brings real-time features to CORBA by specifying a priority-based scheme for handling object requests. RT-CORBA differs from our RT-Llama architecture in that 1) it is deadline-based rather than priority-based and 2) it can reason about the predictability of an entire process rather than just one service at a time. In other words, based on the deadline of the entire process, it manages to determine the intermediate deadline for and schedules each individual service that comprises the process.

Advance reservation systems like that discussed for RT-Llama is a growing research area. For example, Mamat et. al. [14] discuss an advance reservation system for clusters, particularly to help manage I/O conflict among nodes in a cluster. They identify an advance factor for reservations that are multiples of task interarrival times, anywhere from immediate to a factor of ten. However, no special discussion of data structure would be required for such a short time range. Our research, however, requires efficient data structures for storing managing reservations on the order of seconds to minutes in advance.

The GARA system [15] is another advance reservation system that is especially useful for reserving bandwidth for streaming applications. However, streaming media applications have coarser requirements than the individual service execution reservations required for RT-SOA.

VI. Conclusion

SOA has gained wide acceptance in the past few years but due to its unpredictable nature, cannot incorporate real-time applications. In this paper, we attempt to bridge that gap by creating an RT-Llama SOA framework that allows users to specify end-to-end deadlines on their business processes. RT-Llama in turn, after performing initial screening and feasibility checks based on cached utilization data, plans the full BP execution by efficiently reserving resources in advance on the hosts where each service is to execute. We have explored the merits of various reservation mechanisms, demonstrating the flexibility of RT-Llama for different environments.

References

[1] M. Bichler and K.-J. Lin, "Service-oriented computing," *IEEE Computer*, vol. 39, no. 3, pp. 99–101, March 2006.

[2] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, January-February 2005.

[3] K.-J. Lin, M. Panahi, Y. Zhang, J. Zhang, and S.-H. Chang, "Building accountability middleware to support dependable SOA," *IEEE Internet Computing*, vol. 13, no. 2, pp. 16–25, 2009.

[4] Y. Zhang, K.-J. Lin, and J. Y. Hsu, "Accountability monitoring and reasoning in service-oriented architectures," *Journal of Service-Oriented Computing and Applications (SOCA)*, vol. 1, no. 1, 2007.

[5] R. Martin, "Data latency playing an ever increasing role in effective trading," *InformationWeek*, May 2007.

[6] S. Blake, D. Black, M. Carlson, M. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services, RFC 2475*, 1998.

[7] R. Braden, D. Clark, and S. Shenker, *Integrated Services in the Internet Architecture: an Overview, RFC 1633*, 1998.

[8] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[9] L. Abeni and G. Buttazzo, "Resource reservation in dynamic real-time systems," *Real-Time Systems*, vol. 27, no. 2, pp. 123–167, 2004.

[10] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, May 2007.

[11] S. A. Moses, L. Gruenwald, and K. Dadachanji, "A scalable data structure for real-time estimation of resource availability in build-to-order environments," *Journal of Intelligent Manufacturing*, vol. 19, no. 5, pp. 611–622, 2008.

[12] A. Bou-Ghannam and P. Faulkner, "Enable the real-time enterprise with business event processing," *IBM Business Process Management Journa*, no. 1.1, December 2008.

[13] Object Management Group, *RealTime-CORBA Specification, v 2.0*, OMG Document formal/03-11-01 ed., Object Management Group, November 2003.

[14] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling with advance reservation," in *Euromicro Conference on Real-Time Systems, ECRTS '08.*, July 2008, pp. 37–46.

[15] I. T. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, "End-to-end quality of service for high-end applications," *Computer Communications*, vol. 27, no. 14, pp. 1375–1388, 2004.