International Journal of Communication Networks and Security

Volume 1 | Issue 2 Article 1

2011

Dynamic Thread Pool based Service Tracking Manager

D.V. Lavanya
National Institute of Technology Calicut Calicut, India, lavanya.vijaysri@gmail.com

V.K. Govindan
National Institute of Technology Calicut Calicut, India, vkg@nitc.ac.in

Dynamic Thread Pool based Service Tracking Manager

D.V.Lavanya, V.K.Govindan
Department of Computer Science & Engineering
National Institute of Technology Calicut
Calicut, India
e-mail: lavanya.vijaysri@gmail.com, vkg@nitc.ac.in

Abstract:The Service Tracking Manager (STM) application is a management software that integrates, collects, and manages alerts/events information from Casino Management Systems. It allows casinos to track alerts and events at different locations. Casinos staff attends to various kinds of requests. The requests can be for providing customer service, handle events related to slots, alerts for security reasons, and so on. As the size of the casino increases, handling, managing, and prioritizing the requests become difficult and casinos resort to hiring more staff.

STM is independent of the host system and receives requests from Slot Management Systems, Casino Management Systems and other source applications as appropriate. Once the requests are received, the STM application helps in managing and prioritizing the requests. The main functionality or the key operation to be performed on the data is the parsing to check if it is required by the STM system. To distribute the work, the Thread Pool concept is used. The STM application also runs the logic for prioritizing the tasks and assigns it to the staff. The present work proposes a service tracking manager based on dynamic thread pool concept. The paper examines the correlation between the internal characteristics and the performance of the static and dynamic thread pool. The simulation results show that dynamic optimization for thread pool size is very effective in alleviating the management overhead and improving the overall performance.

Keywords: Event handling, Alert monitoring, Thread pool, Static and Dynamic Optimization

I. INTRODUCTION

Casinos have usually staff people attend to various kinds of requests. Requests could be for various reasons. It can be for providing good customer service, handle events related to slots (like, jackpot payout, immediate trouble shooting of slot machine problems), alerts for security reasons. As the size of the casino increases, handling, managing and prioritizing the requests becomes difficult and casinos usually resort to hiring more staff. Proposed product called Service Tracking Manager does help to alleviate the problem. STM sits independent to the host system and receives requests from SMS, CMS and other applications as appropriate [2]. Once the requests are received, it helps in managing and prioritizing the requests. STM application helps in converting the events to tasks based on rules based engine. It also runs logic for prioritizing the tasks and assigns it to the staff. STM application can also be foreseen as a centralized system for handling alerts. Benefits of the STM application are as follows:

- Ability to receive the events or messages or alerts from the host systems.
- Convert the events into tasks by applying rules and heuristics.
- Handle the events/messages/alerts as tasks and display the tasks on handheld terminals.
- Track the performance of the staff.

II. SERVICE TRACKING MANAGER

The proposed Service Tracking Manager is a web based application which creates tasks in real time on a casino floor by intercepting messages from other Casino Management Applications. Once the requests are received, the STM helps in managing and prioritizing the requests. The key operation to be performed on the data is to parse it. Parse the data to check if it is required by the STM system. To distribute the work, Thread pool concept is used. A thread pool is a collection of threads that can be used to perform a number of tasks in the background. This leaves the primary thread free to perform other tasks asynchronously. The proposed system uses dynamic thread pool to improve the performance of the service manager. The STM application converts the events to tasks based on a Rule based engine. It also provides hand held based access to the casino floor personnel. The rest of the sections of this paper are organized as follows: Section III briefly reviews the thread pool concepts used in existing systems. The structure of the Service Tracking manager (STM) is described in Sections IV. The functioning of the major components- data receiver, parser and rule engine of STM is depicted in this Section. Section V discusses the performance evaluation of STM with various thread pool sizes for static and dynamic cases. The performance results are given in Section VI, and finally Section VII concludes the paper.

III. LITERATURE SURVEY

The proposed system converts the events to tasks based on rules based engine. The Data Receiver is the starting point of the STM where it is plugged to an external system which is the source for the events and constantly receives data from it [3]. The messages received obey the Freeform protocol [1]. A thread pool is a collection of threads that can be used to perform a number of tasks in the background. Each incoming request is assigned to a thread from the thread pool, so the request can be processed asynchronously, without tying up the primary thread or delaying the processing of subsequent requests [5].

Multithreading approach is a clean design approach to handling asynchronous requests. A dynamic thread pool model is one of the multithreaded server programming model that handles many requests from users concurrently [4]. The architecture used to implement multithreading can have a large impact on the computational thread creation overhead. Two models including thread-per-request and thread pool are widely used in multithreaded programming for server applications. The thread-per-request model spawns a thread for each request and destroys the thread after finishing the request. In contrast, a thread pool system spawns and maintains a pool of threads. When a request arrives, the application uses a free thread in the pool to serve a client request, and returns the thread to the pool after finishing the request.

Experimental studies suggest that a thread pool model can significantly improve system performance and reduce response time [5]. Because of its benefits, thread pool systems have been adopted by a large number of popular server applications, such as Apache and Windows IIS.

IV. COMPONENTS OF STM

The 3 main components of the STM are

- Data Receiver
- Parser
- Rule Engine

The details of these components of STM are described in the following subsections.

A. Data Receiver

The Data Receiver is started as a windows service which is designed to constantly receive the data from the external system. It uses a common port to which the external system agrees upon to send the data. The data is received through the use of windows sockets programming. The data received from an external system obeys the Freeform protocol. Hence the system needs to be aware of the Freeform protocol so that the data can be obtained and converted to a simple understandable format by all the other STM components [1].

At the core of its architecture, the data receiver is designed to receive the incoming data via Socket Listeners. Socket Listeners will be as many as there are sources to receive the data from. The main functionality or the key operation to be performed on the data is to parse it. Parse the data to check if it is required by the STM system.

To distribute the work, the Thread Pool concept is used. A thread pool is a collection of threads that can be used to perform a number of tasks in the background. This leaves the primary thread free to perform other tasks asynchronously. Once a thread in the pool completes its task, it is returned to a queue of waiting threads, where it can be reused. This reuse enables applications to avoid the cost of creating a new thread for each task.

The description of the Data Receiver classes is as given below:

1) SocketListener: SocketListener creates a TCP/IP socket for listening data. The socket listening for connections is asynchronous type. Whenever a client successfully connects to this socket, the SocketListener instantiates the SocketReceiver class which will receive data from this client. The operations on the received data is all taken care in the SocketReceiver.

2) SocketReceiver: SocketReceiver receives the incoming data on a socket provided by the SocketListener. The data receiving is of asynchronous type. One single instance of a MessageStitcher class is used for every single instance of SocketReceiver. Hence the received incoming data is passed to the MessageStitcher class for further analysis.

3) MessageStitcher: The MessageStitcher class is used to check if the data received from a socket is received completely. Sometimes the socket might receive one complete message in more than one receive operation. In such a case, the MessageStitcher will check for the length of the message (which is generally sent with the message itself) and then checks if the entire message received is actually equal to the length of the message specified in it. If not, then the MessageStitcher will actually wait for the rest of the half message to be received and sent by the SocketReceiver. The MessageStitcher class will then stitch the two sets of data to form a complete message. After stitching the message, it is then sent to the ThreadPooler class which will allocate the tasks to a fixed number of available threads to perform.

4) ThreadPooler: This class initially creates a fixed number of threads (10. This could be increased based on the density of incoming messages. But it should ideally be kept at an exact number so that system does not suffer from far too many threads). Every thread will check for the presence of a task from the common queue. If there is no task to be done by the threads, then the threads go to the wait state until they get signalled. Once a task is arrived, the Thread Pooler will signal the threads that there is a task to be picked up. Suppose if all the threads are busy doing some task, then the Thread Pooler will add it to a common queue since there is no thread free to be signalled. Once a thread is free, it will pick up from the common queue.

B. Parser

A parser breaks data into smaller elements, according to a set of rules (Freeform protocol) [1] that describes its structure. Most data can be decomposed to some degree. The data is received through the Data Receiver. The data received from an internal system obeys the Freeform protocol. Hence the system needs to be aware of the Freeform protocol so that the data can be obtained and parsed to a simple understandable format. The Data Receiver receives the data and processes it. The freeform exceptions collect the raw data and send it to the freeform parser. The Freeform parser decomposes the data based on some set of rules and classify into several set of events. Figure 1 shows its interaction with the Rule Engine.

C. Rule Engine

Once the events come into STM system, STM application picks up for processing if event can be mapped to a preconfigured event category. STM application loops through the configured rules (Rules are attached to predefined event categories). Application shall provide options for the end user to configure rules based on data elements present in STM application. Once a rule matches, STM application stops processing the rules and picks up what action to be taken based on what is configured in the rules. If the action to be taken is not configured in the rules, then action to be taken are driven by what is configured in event category.

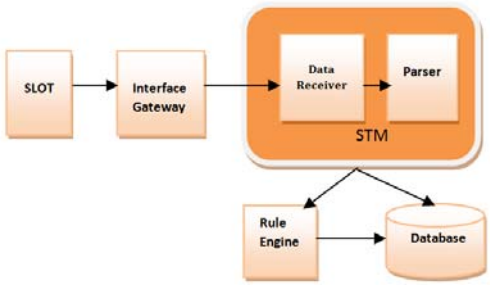

Figure 1: The existence of Parser in STM

STM application helps in converting the events to tasks based on rules fired in the rule engine. All the rules must be configured in the rule engine for a task. For this phase for an event or an alert there could be a one to one mapping of task [2]. When the event comes into STM application it has to be matched with a rule for a task. In case if it doesn't match then it is to be discarded. As flexibility comes into picture for one rule, also there could be multiple events associated. Figure 2 shows the flow chart of how the rules are handled in the rule engine. Once events are received in the STM system, STM application runs rules based logic to convert events to tasks or send e-mails, SMS or to send the information to surveillance monitor. STM application shall have a Rule builder which helps in determining what should be done with the alert.

V. PERFORMANCE METRICS

Experimental studies suggest that a thread pool model can significantly improve system performance and reduce response time. The performance of the server applications rely in part on the throughput which can be delivered by the thread pool.

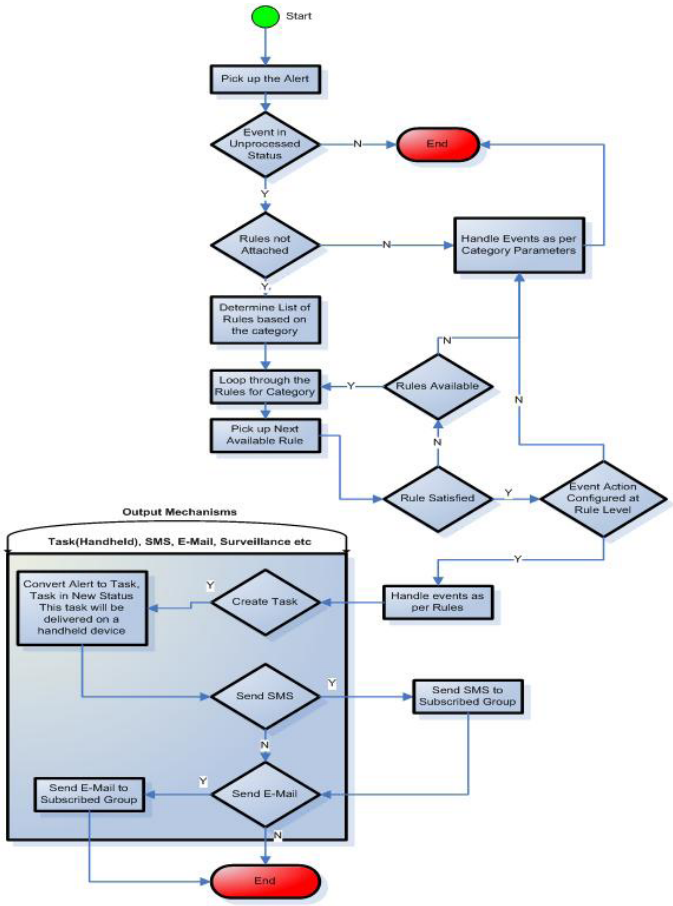

Figure 2: Handling the rules in Rule engine

A major factor which determines the thread pool performance is the pool size. With a larger pool size, the thread pool can handle more tasks simultaneously with faster response time. As the pool size increases, however the overhead of thread pool management will become significant and degrade the system performance eventually. To solve the problem, a dynamic optimization approach which is simpler to avoid huge runtime overhead is required. The dynamic optimization for the thread pool size is very effective in alleviating the management overhead and improving the overall performance.

There are two modes for any thread: busy and idle. At the beginning, all threads are running in idle mode and waiting for the notification of arrivals of new tasks. Whenever new tasks become available, a taskposted signal will be posted. All worker threads waiting for this signal will be notified and compete for a mutex. The winner (called the active thread) will get the mutex and check the pool state. If the task queue is not empty, the active thread will grab one available task in the task queue and run it. The thread pool should also allow users to submit tasks for execution. The functionality is provided by task dispatcher. Specifically, the task dispatcher will put the submitted task into the task queue (shown in Figure 3) and notify the worker threads which are waiting for new tasks. When the task to be done is placed into a queue, the dispatcher function returns immediately. However, the dispatch function has to be blocked when the task queue becomes full.

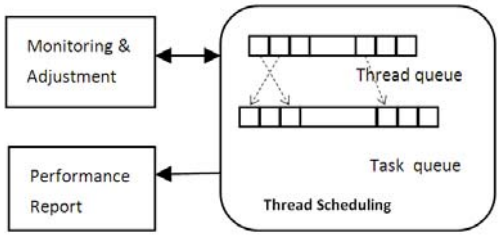

Figure 3: Software Organization of a thread pool System.

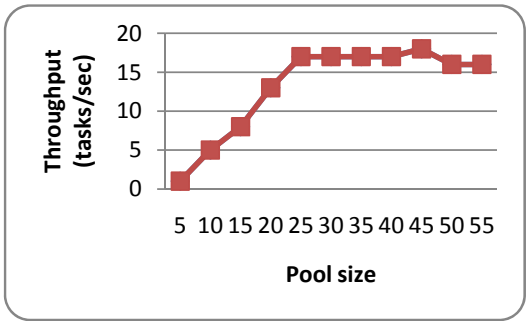

Figure 4: The thread pool management overhead.

VI. RESULTS

The benefit of using a thread pool is to avoid the overhead of thread creation. It does not mean users should create a thread pool as large as possible. Indeed, the overhead for managing threads in the pool can be a big issue. A direct approach for studying this overhead is to observe the throughput while increasing the pool size. The performance improvements brought by a thread pool will be saturated when the pool size reaches a threshold. After that, increasing the pool size will not help to improve the performance further. Instead, the overhead of thread pool management will degrade the performance (throughput) when the size becomes larger. Therefore, by increasing the pool size, we should be able to observe the impact brought by the overhead. Following this idea, we have measured the throughput of our benchmark by increasing the pool size from 5 to 55. The results are presented in Figure 4. According to this figure, the throughput becomes stable when the pool size reaches 25. After that the throughput mostly fluctuates around a fixed value. The best throughput is observed when the pool size reaches 45. After that, it begins to drop gradually.

The point where the throughput becomes stable is called stable point. Beyond this point, the throughput will maintain a relatively steady value. When the size is greater than another threshold, called the degradation point, the overhead of pool management will become dominant and offset the benefits brought by using the thread pool. The performance will decrease after this point. The design of a dynamic thread pool needs to be able to adjust the pool size as quickly as possible to the safe zone, which is the area between the stable point and the degradation point. This is the area where the throughput will reach a maximum without introducing too much overhead.

The other experiment is to compare the throughputs of the static and the dynamic thread pool. The dynamic thread pool is designed to adjust the pool size according to the behaviour of multithreaded applications. The main goal is to achieve better performance without introducing too much overhead. Therefore comparing throughput will help to understand the performance improvement brought by using the dynamic thread pool. The experimental results are shown in Figure 5. To compare the performance in a better fashion, the throughput of the dynamic thread pool is normalized to the throughput of static thread pool. Once events/alerts are received in STM application, STM application runs rules based logic to convert events to tasks or send e-mails, SMS or to send the
information to surveillance monitor. STM application shall have a rule builder which helps in determining what should be done with the alert.

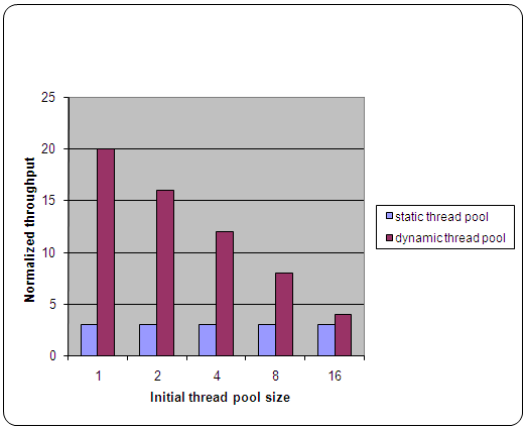

Figure 5: The throughput improvement by using dynamic thread pool.

VII. CONCLUSION

Service Tracking Manager is a proposed solution to handle and prioritize the alerts and events sent to security and floor personnel. The application can be monitored either in the Fixed Station or in the Handheld. The handheld is a device that allows the casino employees to access the events information occurring on the floor through a wireless service. The performance of the system is studied with static and dynamic thread pool cases. The simulation results show that dynamic optimization for thread pool size is very effective in alleviating the management overhead and improving the overall throughput performance.

REFERENCES

[1] Rajiv K.B. "SDS Freeform Messaging Protocol Document" Bally Technologies Internal Document.

[2] Rajiv K.B. "BDS Functional Requirement Document" Bally Technologies Internal Document.

[3] Sunita Vaidya. "BDS User Guide" Bally Technologies Internal Document.

[4] D. Kang, S. Han, S. Yoo and S. Park. "Prediction based Dynamic Thread Pool Scheme for Efficient Resource Usage." IEEE Proceedings on 8th International Conference on Computer and Information Technology - 2004.

[5] Dongping Xu and Brett Bode "Performance Study and Dynamic Optimization Design for Thread Pool Systems" IEEE Proceedings on 8th International Conference on Computer and Information Technology -2003, pp. 54-60.