Complex System Optimization

Subtask of Interactive Service Supporting System in Modern Service Industry

Ronglin Zhang
Abstract

Complex System Optimization

Ronglin Zhang

This report is a work description and summary for complex system optimization which is a sub-task of interactive service supporting system in modern service industry.

There are seven parts in total of the report. The first part is project introduction. It introduces the origin and general background of the project. The second part is a summary of pre-study work, including the pros and cons of existing approaches and techniques to be used in the project. The third part is the system design. It not only gives functional requirements, indicates runtime environment, but also describes in detail the system architecture and several basic but important design ideas. The fourth part is the system implementation. The final system release is also given in the end of this part. Afterwards is conclusion part which sums up the whole work. In the end, there are abbreviations list and references for this report.
Acknowledgements

First I want to thank Xiaomeng Wang for giving me the opportunity to work with this thesis project. I have learned a great deal from this project. I want to thank you for your help throughout the entire project. Your guidance at the beginning, your useful ideas during the project and your final help with the report is greatly appreciated.

I want to thank Zhijiang Fang for all the supervision during the project, which inspired us do our best of best to finish the work.

I want to thank for Olle Eriksson for being my reviewer, shorten my thesis application process.

Finally I want to thank all of my co-workers for giving me much help during the project.
Content

1. Introduction
2. Pre-study
   2.1 Complex system optimization
      2.1.1 Background
      2.1.2 Existing approaches
      2.1.3 Research Orientation
      2.1.4 Techniques
      2.1.5 Research goals
   2.2 Management monitor
      2.2.1 Background
      2.2.2 Research Orientation
      2.2.3 Techniques
      2.2.4 Research goals
3. General Design
   3.1 General functional requirements
   3.2 Runtime environment
   3.3 Basic design ideas
   3.4 System architecture
4. Implementation
   4.1 Key process
   4.2 Log management module
      4.2.1 General design idea
      4.2.2 Implementation
   4.3 User interface module
      4.3.1 General design idea
      4.3.2 Implementation
   4.4 Release
5. Conclusion
6. Abbreviations
7. References
1. Introduction

Nowadays, the global economic tends to change from industrial economy to service economy. More and more manufactories and trading industry depend on resource integration, work flow optimization and cooperation. The modern service industry is taking a bigger and bigger proportion in GDP, and its development has become a symbol of measuring the national modernization level and a commanding height of competition with international industries.

The interactive service supporting system in modern service industry is lead by Ministry of Science and Technology and Ministry of Commerce of the People’s Republic of China. It is an important part of “Modern Service Industry Commonness Supporting System and Applied Demonstration Project” in “National Science and Technology Supporting System”.

It surrounds the requirements of modern service industry development, combines several key techniques in each field, breaks through the integration of service fields, shares the technical problems in cooperation, by integrating government admission, security authentication, criterion information, data integration, electronic payment, statistics management, authentication management, responsibility confirmation, searching service and instant communication and so on such the third commonness services, forms a new service cooperating mechanism.

There are many key techniques related to the project, such as service detection technique based on semantics, loose coupling technique in open environment and large-scale affairs handling and integrating technique and so on. And complex system optimization is one of the indispensable techniques.
2. Pre-study

2.1 Complex system optimization

2.1.1 Background

The modern service industry is mainly oriented to provide services to the public. It has a huge number of consumers. Because of the unpredictable access burst of numerous system users, it is usually that lots of requests get together in a very short time, which directly lowers the whole system performance. Therefore, it is necessary for a service system to support high concurrency, large throughput, robust loading and good response ability for peak value.

2.1.2 Existing approaches

There are two existing approaches:

(a) Thread pool model
In the thread-per-request model, every incoming request causes a new job thread to be spawned to process it. In the thread pool model, the system maintains a pool (or pools) of pre-spawned job threads. When a new request arrives, the request is passed to a free job thread in the pool. After the request is processed, the thread is returned to the pool to execute another request. The advantages of the thread-pool model include that the system can have a bound on the number of threads and reduce the cost of thread creation. The thread pool model is often used in embedded systems and web servers. However, it’s not suitable for high concurrency environment and its loading robustness is poor.

(b) Event-based architecture
The event-driven architecture defines a methodology for designing and implementing applications and systems in which events transmit between loosely coupled software components and services. An event-driven system is typically comprised of event consumers and event producers. Event consumers subscribe to an intermediary event manager, and event producers publish to this manager. When the event manager receives an event from a producer, the manager forwards the event to the consumer. If the consumer is unavailable, the manager can store the event and try to forward it later. This method of event transmission is referred to in message-based systems as store and forward. Building applications and systems around an event-driven architecture allows these applications and systems to be constructed in a manner that facilitates more responsiveness, since event-driven systems are, by design, more normalized to unpredictable and asynchronous environments. There are several benefits of event-driven design and development:

- (1) allows easier development and maintenance of large-scale, distributed applications and services involving unpredictable and/or asynchronous occurrences;
- (2) allows new and existing applications and services to be assembled, reassembled, and reconfigured easily and inexpensively;
- (3) promotes component and service reuse, therefore enabling a more agile and bug-free development environment;
- (4) Short-term benefits: allows easier customization because the design is more responsive to dynamic processes;
(5) Long-term benefits: allows system and organizational health to become more accurate and synchronized closer to real-time changes. However, it does have drawbacks as well - not sufficiently use the multiple thread mechanism and it lacks of the support of operating system, having implementation difficulty.

2.1.3 Research orientation

The research is oriented to design a complex system service optimization framework basing on multiple staged event-driven and achieve high performance platform on system level, service level and coding level and so on.

2.1.4 Techniques

The included two techniques are
(a) Multiple thread scheduler based on SEDA framework
SEDA is an acronym for Staged Event-Driven Architecture, and refers to an approach to software design that decomposes a complex, event-driven application into a set of stages connected by queues. This architecture avoids the high overhead associated with thread-based concurrency models, and decouples event and thread scheduling from application logic. By performing admission control on each event queue, the service can be well-conditioned to load, preventing resources from being overcommitted when demand exceeds service capacity. SEDA employs dynamic control to automatically tune runtime parameters (such as the scheduling parameters of each stage) as well as to manage load (for example, performing adaptive load shedding). Decomposing services into a set of stages also enables modularity and code reuse, as well as the development of debugging tools for complex event-driven applications.

(b) Intelligent efficiency bottleneck detection and dynamic resource arrangement technique - utilizing control strategy combined by global arrangement and self-feedback to arrange the resource efficiently.

2.1.5 Research goals

The research goals are
(a) System optimization of both throughput and latency;
(b) Throughput invariance after reaching the peak value of system requests; fair latency, increases linearly with increase of requests.

2.2 Management monitor

2.2.1 Background

The interactive service supporting system integrates the services from the third party. Security and management become the two problems of the system. Therefore, remote controlling, exception diagnosis and automatic analysis techniques under distributed and heterogeneous system are needed to systematically monitor the system and the corresponding services.
2.2.2 Research Orientation

The research is oriented to design a unified system monitor which can control and manage the system the corresponding services in a consistent way, making the system stable and reliable.

2.2.3 Techniques

The included techniques are
(a) Remote controlling under distributed and heterogeneous system;

(b) Exception diagnosis and automatic analysis;

(c) System management framework based on JMX;

The JMX specification defines architecture, the design patterns, the APIs, and the services for application and network management and monitoring in the Java programming language.

Using JMX technology, a given resource is instrumented by one or more Java objects known as Managed Beans, or MBeans. These MBeans are registered in a core managed object server, known as an MBean server that acts as a management agent and can run on most devices enabled for the Java programming language. The specifications define JMX agents that users use to manage resources instrumented in compliance with the specifications. A JMX agent consists of an MBean server, in which MBeans are registered, and a set of services for handling MBeans. In this way, JMX agents directly control resources and make them available to remote management applications. The way in which resources are instrumented is completely independent from the management infrastructure. Resources can therefore be rendered manageable regardless of how their management applications are implemented. JMX technology defines standard connectors (JMX connectors) that allow users to access JMX agents from remote management applications. JMX connectors using different protocols provide the same management interface. Hence a management application can manage resources transparently, regardless of the communication protocol used. JMX agents can also be used by systems or applications that are not compliant with the JMX specification but which support JMX agents.

JMX technology provides Java developers across all industries with a flexible means to instrument Java code, create smart Java agents, implement distributed management middleware and managers, and smoothly integrate these solutions into existing management and monitoring systems. It has following features:

(1) Enables Java applications to be managed without heavy investment: A JMX technology agent can run on most Java technology-enabled devices, thus Java applications can become manageable with little impact on their design. A Java application simply needs to embed a managed object server and make some of its functionality available as one or several managed beans (MBeans) registered in the object server; that is all it takes to benefit from the management infrastructure.

(2) Provides a standard way to manage Java technology-based applications, systems, and networks. For example, the Java 2 Platform, Enterprise Edition (J2EE) 1.4 Application Server conforms to the JMX architecture and consequently can be managed using JMX technology.

(3) Can be used for out-of-the-box management of the JVM. The Java Virtual Machine (JVM) is highly instrumented using JMX technology. Users can easily start a JMX agent to access the built-in JVM instrumentation, and thereby monitor and manage the JVM remotely.
(4) Provides a scalable, dynamic management architecture. Every JMX agent service is an independent module that can be plugged into the management agent, depending on the requirements. This component-based approach means that JMX solutions can scale from small-footprint devices to large telecommunications switches and beyond. The JMX specification provides a set of core agent services. Additional services can be developed and dynamically loaded, unloaded, or updated in the management infrastructure.

(5) Leverages existing standard Java technologies. Whenever needed, the JMX specification references existing Java specifications, for example, the Java Naming and Directory Interface (JNDI).

(6) Integrates easily with existing management solutions and emerging technologies. For example, JMX agents could be managed through an HTML browser. The JMX APIs are open interfaces that any management system vendor can leverage. JMX solutions can use lookup and discovery services and protocols such as Jini™ network technology and the Service Location Protocol (SLP).

(d) SOAP message transaction;
SOAP stands for Simple Object Access Protocol. It is a communication protocol for communication between applications. It is also a format for sending messages, designed to communicate via Internet. It’s platform-independent, language-independent, based on XML, simple and extensible. It allows users to get around firewalls.
It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

(e) JMS message transport techniques.
The Java Message Service is a Java API that allows applications to create, send, receive, and read messages. Designed by Sun and several partner companies, the JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations. It minimizes the set of concepts a programmer must learn to use messaging products but provides enough features to support sophisticated messaging applications. It also strives to maximize the portability of JMS applications across JMS providers in the same messaging domain.
The JMS API enables communication that is not only loosely coupled but also (1) asynchronous. A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them. (2) Reliable. The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

2.2.4 Research goals

The research goals are
(a) Providing system warning;
(b) Providing node monitor and maintenance;
(c) Providing visualization and management of network topology;
(d) Providing plug-and-play of service modules
(e) Providing powerful operation management service SERVLET;
(f) Providing diverse approaches communicating with system nodes, such as HTTP, SNMP, JMS and so on.
3. General Design

3.1 General functional requirements

The complex system optimization framework divides the applications into several stages. The stages are comparatively independent and connected by event queue. The framework uses dynamic resource control to make applications adaptive to load changes and resource limitations. The functional modules should consider the following factors.

1. High concurrency for event-driven - in order to support higher concurrency, the complex system optimization framework employs event-driven technique to describe multiple flows through the system. This design only uses several threads instead of threads per event.

2. Dynamic thread pool – to reduce the strictness of scheduling and satisfy the non-blocking requirement, the complex system optimization framework uses thread pool to drive actions. It not only makes developers feel freer to handle events scheduling, but also allows event handling application to be blocked in a short time when stages assign other threads.

3. Structured code modules and loading management – by decomposing services or applications into several stages, application developers can focus on the logic and concurrency management of each stage, then combine each stage into a complete service. Event queue separates executions of each stage, allowing stages to be independently developed. Event queue provides a control point in request flow, so that request flow can be detected and managed by application. Similarly, based on each stage, inflow control is executed.

4. Self-adaptive of resource management – the complex system optimization framework doesn’t have to calculate how much resource the application might require in advance, and users’ loading features. It utilizes feedbacks and control to automatically adjust parameters that resource uses. For example, the system determines the number of threads assigned by each stage, instead of setting fixed thread number by developers or administrators during programming.

Based on above, the general functional requirements of the complex system optimization framework are designed as follows.

1. System management module.
   System management module manages stages by invoking stage factory module, including creating and destroying stages, managing stage thread pool and dynamically generating stage graph. Meanwhile, it communicates with resource analysis module, collects related performance data of stages and executes corresponding operations according to the stage control strategy sent by resource analysis module. In addition, it generates some basic stages for users to choose by common stage library module. These stages encapsulate some bottom layer functions.

2. Stage factory module
   Stage factory takes the responsibility of creating and assembling components in each stage. It receives information about parameters from system management module, creates components with different functions based on different strategies according to parameters, and assembles
them into one complete stage. What’s more, when stages are running, each stage is controlled by system management module and gives related resource information to system analysis module.

System analysis module analyzes the usage of resources when each stage operating, adjusts overall resources based on strategies, making the system maintain high performance. It receives loading information from each stage and collects related overall resource information of the system, and then adjusts overall system performance according to corresponding strategy. The detailed adjustments are executed by system management module.

Common stage library provides implements of most of lower-layer stages, including NBIO, asynchronous socket, HTTP protocol library, SSL/TSL protocol library and so on, providing convenience during the development of the framework.

3.2 Runtime environment
Operating system: Linux, Windows XP
Unified modeling tool: StarUML
Programming tool: Eclipse 3.3, JDK1.5

3.3 Basic design idea
The basic design idea of the system is dividing client applications into a set of interdependent stages. Each stage completes an independent logical function. They communicate with each other by events. Stage receives event sent from other stage in the event queue. The inner thread pool implements transaction of event concurrency. Each stage controls resource management itself, including adjustment of thread pool, controlling event enqueuing and adjustment of batch processing and so on. In addition, implementing overall management of stage thread pool can generally increase its utilization ratio. Design of each stage is as follows.

The components include:

![Design of each stage](image)
(a) Event queue;  
(b) Event handler;  
(c) Thread pool;  
(d) Controller.

The system main instance reads information from configuration when system starts, creates each stage instance and constructs stage graph. Main instance, in the mean time, needs to start thread pool of each stage. After starting successfully, it needs to start system thread pool management instance, resource analysis instance and stage graph display control instance. The system thread pool management instance assigns each stage the number of thread resource. The resource analysis instance collects resource information and makes corresponding analysis and decision. The stage graph display control instance displays stage graph and dynamically updates when stage graph has any change. Main instance monitors these three instances and restarts certain instance if it stops exceptionally.

![Figure 3 System instances distribution](image)

1. Main instance read configure information from files. The system uses XML file recording configure information, utilizes JDOM to interpret files and generates corresponding data objects.

2. Main instance communicates with stage graph display control instance by message event. Main instance notifies stage graph display control instance by sending event after it changes stage graph. Stage graph management instance starts drawing after it gets stage graph handle. Here relates to two threads’ operations to one data structure, so interlocking needs to be added to synchronization.

17
3. Thread pool management instance maintains one event queue. Each stage’s thread pool notices the instance to adjust thread resource by inserting event into the queue.

4. There should be asynchronous socket based on operating system bottom layer non-blocking I/O mechanism and files operations. System encapsulates these functions into a group of common stages, dispatches I/O requests and events to stages to handle. Application uses asynchronous I/O by adding stage into the group.

5. Java application is used for interpretation and therefore its implementation efficiency is not high as C, but using JIT can improve this.

6. The system log function is implemented by Common-logging. The Logger at lower level employs Log4J.

**3.4 System Architecture**

![System Architecture Diagram]

The system is consisting of the following modules.
(a) System management module;
(b) Stage factory module;
(c) Resource analysis module;
(d) Common stage library module.
Each module’s functions and relationship with other modules is displayed as following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
<th>Interact with</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System management module</strong></td>
<td>1. stage management;</td>
<td>1. Stage factory: system management module creates stage instance by calling factory functions, it also provides corresponding configuration parameters.</td>
</tr>
<tr>
<td></td>
<td>2. stage graph management;</td>
<td>2. System analysis module: system management module passes initiation parameters to system analysis module. System analysis module sends stage feedback. According to the feedback information, system management module adjusts stages and stage graph.</td>
</tr>
<tr>
<td></td>
<td>3. system configuration;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. resource control;</td>
<td>3. Common stage library module: common stage library module is used for system management module to create a group of common stages and save in the common stage list. When application uses the stage in the list, system management module then initiates the stage.</td>
</tr>
<tr>
<td></td>
<td>5. log management;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. interaction with users</td>
<td></td>
</tr>
<tr>
<td><strong>Stage factory module</strong></td>
<td>1. stage assembling;</td>
<td>1. System management module: when assembling stages, stage factory module receives parameters from system management module. When stages running, stages receive control information from system management module.</td>
</tr>
<tr>
<td></td>
<td>2. event queuing and dequeuing management;</td>
<td>2. Resource analysis module: when stages running, each stage sends loading feedback to resource analysis module.</td>
</tr>
<tr>
<td></td>
<td>3. event handling;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. thread pool management;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. stage latency control;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. stage running and maintenance;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. stage status maintenance</td>
<td></td>
</tr>
<tr>
<td><strong>Resource analysis module</strong></td>
<td>1. overall loading monitor;</td>
<td>1. Stage factory module – resource analysis module receives loading information from each running stage’s feedback;</td>
</tr>
<tr>
<td></td>
<td>2. overall loading adjustment</td>
<td>2. System management module: resource analysis module sends stage adjustment request to system management module.</td>
</tr>
<tr>
<td><strong>Common stage library module</strong></td>
<td>1. asynchronous socket;</td>
<td>1. System management module: common stage library module is initialized by system management module, and creates stages by it as well.</td>
</tr>
<tr>
<td></td>
<td>2. asynchronous file I/O</td>
<td></td>
</tr>
</tbody>
</table>

(a) The system management module consists of 6 sub-modules.
1. stage management;
2. stage graph management;
3. configuration;
4. resource control;
5. log management;
6. user interface

Figure 5 Sub-modules of system management module

Each sub-module’s functions and relations with other sub-modules are described in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
<th>Interact with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration sub-module</td>
<td>1. read configure information from files and interpret;</td>
<td>1. Stage management sub-module: stage management sub-module gets related</td>
</tr>
<tr>
<td></td>
<td>2. create system configuration data and description of stages</td>
<td>configure information from configuration sub-module and creates stage instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Log management sub-module: log management sub-module gets initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters from configuration sub-module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Resource control sub-module: resource control sub-module gets initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters from configuration sub-module.</td>
</tr>
<tr>
<td>Stage management sub-module</td>
<td>1. create stages; 2. destroy stages; 3. decompose stages; 4. copy stages; 5. merge stages</td>
<td>1. Configuration sub-module: get related configure information from configuration sub-module and then creates stage instance. 2. Log management sub-module: stage management sub-module calls log recording interface in log management sub-module and records corresponding log information. 3. Stage graph management sub-module: stage graph management sub-module gets stage information, creates stage graph, and dynamically updates the graph according to the stage information. 4. Resource control sub-modules: resource control sub-module manages overall thread pool of stages.</td>
</tr>
<tr>
<td>Stage graph management sub-module</td>
<td>1. create stage graph; 2. update stage graph in real time</td>
<td>1. Stage management sub-module: stage graph management sub-module gets stage information from stage management sub-module. 2. Log management sub-module: stage graph management sub-module calls log recording interface in log management sub-module and records corresponding log information. 3. User interface sub-module: user interface sub-module gets stage graph information from stage graph management sub-module and draws stage graph on user graphical interface.</td>
</tr>
<tr>
<td>Resource control module</td>
<td>1. adjust and control overall thread pool of stages</td>
<td>1. Stage management module: resource control sub-module manages overall thread pool of stages. 2. Log management sub-module: resource control sub-module calls log recording interface in log management sub-module and records corresponding log information.</td>
</tr>
<tr>
<td>Log management sub-module</td>
<td>1. initialize parameters; 2. record log</td>
<td>1. Configuration sub-module: log management sub-module gets initialization parameters from configuration sub-module. 2. Stage management sub-module: stage management sub-module calls log recording interface in log management sub-module and records corresponding log information. 3. Stage graph management sub-module: stage graph management sub-module calls log recording interface in log management sub-module and records corresponding log information.</td>
</tr>
</tbody>
</table>
| User interface sub-module | 1. commandline interface;  
2. graphical interface |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Resource control module: resource control sub-module calls log recording interface in log management sub-module and records corresponding log information.</td>
<td></td>
</tr>
<tr>
<td>5. User interface sub-module: user interface sub-module calls log recording interface in log management sub-module and records corresponding log information.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage graph management sub-module</th>
<th>user interface sub-module gets stage graph information from stage graph management sub-module and draws stage graph on user graphical interface.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log management sub-module: user interface sub-module calls log recording interface in log management sub-module and records corresponding log information. Users can view log content through graphical interface.</td>
<td></td>
</tr>
</tbody>
</table>

(b) The stage factory module consists of 9 sub-modules.  
1. stage assembler;  
2. event queue;  
3. event handler;  
4. dynamic thread pool;  
5. stages;  
6. batch processing controller;  
7. thread pool controller;  
8. queue controller;  
9. feedback controller
Figure 6 Sub-modules of stage factory module

Each sub-module’s functions and relations with other sub-modules are described in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
<th>Interact with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage assembler sub-module</td>
<td>1. create complete stage</td>
<td>1. Stages sub-module: according to configuration parameters, stage assembler creates a stage and gets its reference.</td>
</tr>
<tr>
<td>Module</td>
<td>Actions</td>
<td>Module</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>Stages sub-module</strong></td>
<td>1. create and manage event handler;</td>
<td>1. Event handler sub-modules:</td>
</tr>
<tr>
<td></td>
<td>2. create and manage dynamic thread pool;</td>
<td>2. Dynamic thread pool sub-module:</td>
</tr>
<tr>
<td></td>
<td>3. create and manage event queue;</td>
<td>3. Event queue sub-module:</td>
</tr>
<tr>
<td></td>
<td>4. create and manage feedback controller;</td>
<td>4. Feedback controller sub-module:</td>
</tr>
<tr>
<td></td>
<td>5. provide universal managing interface for system management module;</td>
<td>5. Event queue sub-module:</td>
</tr>
<tr>
<td></td>
<td>6.</td>
<td>6.</td>
</tr>
<tr>
<td><strong>Event queue sub-module</strong></td>
<td>1. maintain a group of unhandled events based on strategies;</td>
<td>1. Stages sub-module: event queue</td>
</tr>
<tr>
<td></td>
<td>2. provide event enqueuing and dequeuing;</td>
<td>2. Dynamic thread pool sub-module:</td>
</tr>
<tr>
<td></td>
<td>3. maintain status information of event queue</td>
<td>3. Event queue sub-module:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Batch processing controller sub-module:</td>
</tr>
<tr>
<td><strong>Event handler sub-module</strong></td>
<td>1. handle events</td>
<td>5. Queue controller sub-module:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic thread pool sub-module</strong></td>
<td>1. create, delete and manage threads;</td>
<td>1. Stages sub-module: according to configuration parameters, event handler</td>
</tr>
<tr>
<td></td>
<td>2. thread in the pool dequeues the event from event queue, passes it to</td>
<td>2. Dynamic thread pool sub-module:</td>
</tr>
<tr>
<td></td>
<td>event handler to handle, and enqueues the finished event into next event</td>
<td>3. Event queue sub-module:</td>
</tr>
<tr>
<td></td>
<td>queue</td>
<td>4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-module</td>
<td>Control Event Enqueuing</td>
<td>Confirm the Number of Dequeued Events</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Queue controller sub-module</td>
<td>1. control event enqueuing</td>
<td>1. confirm the number of dequeued events</td>
</tr>
<tr>
<td>Batch processing controller sub-module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feedback controller sub-module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c) Resource analysis module consists of 5 sub-modules:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. resource collection;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. performance analysis;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. key route analysis;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. stage optimization;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. date record</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Each module’s functions and relationship with other modules is displayed as following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
<th>Interact with</th>
</tr>
</thead>
</table>
| Resource collection sub-module | 1. collect system loading information;  
                                  2. collect loading feedback from each stage | 1. Performance analysis sub-module: resource collection sub-module provides data information about stage performance for performance analysis sub-module;  
                                  2. Data record sub-module: resource collection sub-module records data information through data record sub-module. |
| Performance analysis sub-module | 1. detect performance bottleneck;  
                                  2. predict performance bottleneck | 1. Resource collection sub-module: ;  
                                  2. Stage optimization sub-module: performance analysis sub-module sends the result of bottleneck analysis to stage optimization sub-module. |
| Key route analysis sub-module | 1. solve the key route of stage graph         | 1. Stage optimization sub-module: key route analysis sub-module sends the result of key route analysis to stage optimization. |
| Stage optimization sub-module | 1. analyze the results of performance and key route analysis; 2. adjust strategies based on optimization algorithm | 1. Performance analysis sub-module: performance analysis sub-module sends the result of bottleneck analysis to stage optimization sub-module. Stage optimization sub-module then analyzes the received information and dynamically optimizes one or more stages based on optimization algorithm. 2. Key route analysis sub-module: key route analysis sub-module sends the result of key route analysis to stage optimization. Stage optimization sub-module then analyzes the received information and dynamically optimizes one or more stages based on optimization algorithm. |
| Data record sub-module | 1. record loading information | 1. Resource collection sub-module: resource collection sub-module records data information through data record sub-module |
4. Implementation and evaluation

4.1 Key process

The system management module shall have the following key processes.
1. start system;
2. stop system;
3. assign thread pool resource;
4. get stage information;
5. display stage graph

1. Starting system
When users start the system either by command or graphical interface, the system start process starts and initializes the system.
2. **Stop system**
This is the inverse process of starting system. Users stop the system by command or graphical interface. Stages stop running and release all resources.
3. assign thread pool resource
System thread pool manager adjusts thread pool resources, according to the requests from stage’s thread pool, assigns resource fairly.
4. **get stage information**
Users can inquiry stage information by command or graphical interface.
5. display stage graph
System dynamically displays stage logic topological graph via graphical interface.

![Figure 12 Steps of inquiring stage information](image)

**Start**

- Inquiry stage information
- Get stage information
- Send back information list to user interface
- User interface displays stage information

**Stop**

![Figure 13 Steps of displaying stage graph](image)

**Start**

- Stage graph display control thread reads stage information
- Stage graph display control thread draws stage graph
- Update stage graph when stage graph changes

**Stop**
The common stage library module shall have the following key processes.
1. asynchronous socket library initialization;
2. asynchronous socket library I/O request;
3. asynchronous file I/O library initialization;
4. asynchronous file library I/O request

1. asynchronous socket library initialization
When system starts, asynchronous socket layer should be initialized.

![Diagram of asynchronous socket library initialization steps]

2. asynchronous socket library I/O request
Network request should be handled.
3. asynchronous file IO library initialization
When system starts, asynchronous file the IO layer will be initialized.

---

Figure 15 Steps of handling asynchronous socket library I/O request

Start

1. Network request be passed to asynchronous socket library manger
2. Enqueue the request to stage queue
3. Stage handles request
4. Return result to upper application layer

Stop

---

Figure 16 Steps of asynchronous file IO library initialization

Start

1. Initialize asynchronous file IO manager
2. Initialize asynchronous file IO thread manager
3. Initialize asynchronous file IO request handling stage

Stop
4. asynchronous file library I/O request
Disk file request shall be handled.

Start

Asynchronous file I/O manager enqueues I/O request to I/O request handling stage queue

I/O request handling stage handles I/O request

Stop

Figure 17 Steps of handling asynchronous file library I/O request

The system error handling is related to these modules: system management module, stage factory module, resource analysis module and common stage library module. The design principles include throwing and printing exceptions, recording logs, exiting system and handing in to upper layer to make default actions.

The system maintenance is aimed to optimize system performance, ease system maintainers to maintain the system and for users to make a better use of the system. The design principles include maintaining the system according to log records, training system administrators and users, maintaining log records, and system backup.

4.2 Log management module

4.2.1 General design idea

Generally speaking, there usually is 4% of code used for maintaining system log. Log plays a very important role in the whole system, providing important reference for system maintenance and debugging. In this system almost all modules have interactions with log management module. Therefore, for log management module, the design principles should concern much about flexibility, structure and encapsulation and efficiency. We decide to combine the Apache open source project log4j and common-logging approach. Log4j is a product of Apache Software Foundation. With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary. Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and
confidence in the application. Logging and testing should not be confused. They are complementary. When logging is wisely used, it can prove to be an essential tool. One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps reduce the volume of logged output and minimize the cost of logging.

\[\text{LogMessage} \rightarrow \text{writeLog()}\]

\[\text{SubSysException} \rightarrow \text{writeLog()}\]

\[\text{LogMgr}\]

+string className
+String methodName
+main()
+writeLog()

\[\text{log4j common-logging}\]

4.2.2 Implementation

1. Download jar package, add toclasspath: log4j-1.2.14.jar and commons-logging.jar;
2. Write commons-loggin.properties property file, specify logger, set log level for specified logger, output format and output location and so on;
3. Set log4j property file, mainly include the levels of log – OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, output location by specifying appendName and so on;
4. Write log handling class, provide invoke interface for other modules.

A simple log application:
Configuration file:

```
# config root logger
log4j.rootLogger = INFO, system.out
log4j.appender.system.out = org.apache.log4j.ConsoleAppender
log4j.appender.system.out.layout = org.apache.log4j.PatternLayout
log4j.appender.system.out.layout.ConversionPattern =this4j-->%5p {%F:%L} - %m%n

# config thisProject.file logger
log4j.logger.thisProject.file = INFO, thisProject.file.out
log4j.appender.thisProject.file.out = org.apache.log4j.DailyRollingFileAppender
log4j.appender.thisProject.file.out.File = logContentFile.log
log4j.appender.thisProject.file.out.layout = org.apache.log4j.PatternLayout

Application:
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.Logger;
```
public class UserLog4jWithJCreate
{
    public UserLog4jWithJCreate()
    {
        PropertyConfigurator.configure("log4j.properties");
        Logger loggerConsole = Logger.getRootLogger();
        loggerConsole.info;
        loggerConsole.warn;
        Logger loggerFile= Logger.getLogger("thisProject.file");
        loggerFile.warn;
    }
    public static void main(String[] args)
    {
        UserLog4jWithJCreate mainObj=new UserLog4jWithJCreate();
    }
}

4.2 User interface module

4.2.1 General design idea

Stage graph is mainly used for describing the organization and control relation within stages by graphical pattern. It consists of stage nodes and directed edges. The edges come out from sending stage to receiving stage. There are four interactive classes in the module. StageMgr class is responsible for maintaining application stage and manages stage graph. UserInterface class is used for displaying stage graph, handles user requests and updates stage graph when needed. SinkProxy class takes in charge that when stage changes, it records the changed information into data structure, and passes parameters to StageMgr class. StageGraphEdge class is a data structure class defining the data structure information of stage graph. These four classes have the following interaction.
4.2.2 Implementation

As shown in the structure above, the implementation is mainly implemented in UserInterface class. It reads data structure of stage when needed and finishes drawing in the user interface. When stage changes, it can also update the stage graph. Communication with outer modules is implemented by message mechanism. The main thread of UserInterface class receives message. The IDE of whole system is Eclipse, so the implementation is achieved by combination of SWT, draw2D and Java Swing. SWT and draw2D have very good features for drawing good graphics; can make up the drawbacks of Java Swing, and simplify basic drawing, reduce dependency on third tools. They also have good encapsulation, such as org.eclipse.draw2d.graph which provides some important classes e.g. DirectedGraph, Node, Edge, NodeList and EdgeList, which help a lot on creating diagrams. Stage graph is a vector graphic as well which is favorable for drawing vector graphics. This is an example of vector graphic drawn by SWT, draw2D.

Drawing steps are:
1. Generate data from XML file or other data source such as stage data structures defined by developers;
2. Zoom in and out - calculate coordinates, generate x axis and y axis from given data, correspond points from data source with points in the graph;
3. Fill in data structures of Node and Edge;
4. Draw DirectedGraph graph using Eclipse plug-in, such as using drawDotsAndConnections(), drawNode() and drawEdge() to draw node edges.

The data structure definition of stage graph is basically corresponding with Node and Edge in DirectedGraph, as follows:
Vector Stages: vector table type, store stage’s object information (object handle) in stage graph;
Vector Edges: vector table type, store stage’s edge information (object handle) in stage graph, including start stage and stop stage.

4.4 Release

The user interface module is used for better interaction with users and an easier use of the system. The module provides the following functions: displaying stages’ dynamic information, displaying stages’ relations, displaying system performance and displaying system log information in four pages respectively. The system main window is shown as follows:

1. Basic functions
   (1) Stop system
       Clicking red button can stop the whole system, but not close the window. Clicking “Exit” in the “File” menu can shut down the whole system and the window.
   (2) Start system
       Clicking the blue button can start the whole system.
   (3) Close window
       Clicking the third button can close the window but not exit system.
   (4) System tray
       There is a tray function like MSN. Clicking “Close” button, the system icon will appear in the taskbar.
Left-clicking the icon will display the system information. “The Sysopti system is running.”

Right-clicking the icon will display the menu.

2. System monitor
The first page displays the system performance, meaning generally CPU, memory and memory usage of Java Virtual machine.

(1) CPU monitor;
It monitors CPU performance. On left side there is a widget, expresses a percentage of memory usage of CPU. On right side there is a CPU curve that indicates the utilization ratio of each CPU. Since the test environment only has one CPU, so it displays the performance of CPU No.0.
(2) Memory monitor;
It monitors the memory performance. On left side is a widget that indicates the percentage of memory usage. On right side is a memory curve.

(3) Java Virtual Machine monitor;
The curve in the graph indicates the usage of JVM. The first line says the used memory of JVM. The second line is the total memory size that JVM can use.
3. Stage monitor;
The second page displays stage monitor, providing observations of stages.

- **Stage monitor**
- **Stage graph**
- **System monitor**
- **Log management**

**Operations**
- Start stage observation
- Stop stage observation
- Data length
- Update frequency
- Stage data list
- Add stage
(1) Stage monitor
In the stage monitor page, users can add stage to be observed. The added stage displays in small graph. The following is information in the stage data list.

This is the graph after adding a stage.

From the drop-down menu, users can continue to add stage. Each stage will be displayed in a small graph as shown as follows.
Click the first button “Start stage observation” to start observation.

There might be a prompt window as follows that indicates users to input data length between 1 to 50. The sentence in the window is “Please input data length between 1 to 50!”

When the data is correct, click “Start stage observation” button, the graph will display the following information.
When user wants to observe certain stage, click on the small graph, it will be zoomed in.

(2) Stop monitor
Click the second button “Stop monitor button”, the stage observation will stop. Click “Start stage observation”, it will start again.
(3) Set update frequency
Select the update frequency from drop-down menu.

(4) Peak value reminder
On up-right side there will be a peak value reminder reminds the new value exceeds the history peak value.

4. Stage graph
The third page displays the relations among stages, transfer directions of events or requests. The layout of stage graph is automatic.
The following table displays properties of each stage.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Status</th>
<th>Benign</th>
<th>Permit Queue</th>
<th>Queue Size</th>
<th>Threshold Value</th>
<th>Revert to</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>Running</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>LinkedBlockingQ</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GenStage</td>
<td>Running</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>LinkedBlockingQ</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Sink</td>
<td>Running</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>LinkedBlockingQ</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

(2) Drag function
Users can drag stage graph as they want.
5. Log management
The fourth page displays log information.

(1) Display
By default system displays the latest log.

(2) Open other log
(3) Search key word
Users can search key word in the log file.

The search window is similar to that in WordPad in Windows OS.
(4) Set font
Users can set text font, font color, font size and so on.
(5) Edit note
Users can insert note in the log.
Monitor log

If log file has changes, then the new content will be added to the end of log file.

This line starts ThreadPoolController:

Start ThreadPoolController

Start ThreadPoolController

INFO] 2008-01-02 12:01:05,343 method=com.cn.service.keytech.sysopt.sysadmin.threadadmin.Ti
ThreadPoolController : )

INFO] 2008-01-02 12:01:05,343 method=com.cn.service.keytech.sysopt.sysadmin.threadadmin.Ti
ThreadPoolController : )

INFO] 2008-01-02 12:01:05,375 method=com.cn.service.keytech.sysopt.sysadmin.threadadmin.Ti
ThreadPoolController : )
5. Conclusion

The goal of this project is applied to, in the interactive service supporting system, the flow execution engine with long transaction process function and message service bus system; applied to service combination with flow emulating function and flow configuration tools, service flow analysis and optimization tools, dynamic service management system and so on.

In the project start, a pre-study is conducted regarding the subject discussed in this report. The pre-study contains two parts. First part is to discuss how complex system optimization shall be designed and implemented. The background gives the developing orientation. And several pros and cons of existing approaches are also introduced. The techniques of the system are finally determined as (1) Multiple thread scheduler based on SEDA framework - Staged Event-driven Architecture, which decomposes services into a set of comparatively independent stages whereas connected by queues; it employs dynamic control to automatically tune runtime parameters and manage load by using adaptive overload control; (2) Intelligent efficiency bottleneck detection and dynamic resource arrangement technique - utilizing control strategy combined by global arrangement and self-feedback to arrange the resource efficiently.

The second pre-study part is about management monitor. After discussion on background, research orientation and so on, the techniques used in the project are (1) Remote controlling under distributed and heterogeneous system; (2) Exception diagnosis and automatic analysis; (3) System management framework based on JMX; (4) SOAP message transaction.

After the pre-study, a general design of the whole system is created before the actual implementation. The design clearly indicates the data structure and software framework of the system. In this phase the main idea is convert a software requirement into a software expression. This expression is just a general picture of software. Because of the limitation of space, the key data structures are not included in the design. But they have detailed the software picture more specifically and are very close to the final expression of software source code, which helps a lot on code standardization.
6. Abbreviations

GDP – Gross Domestic Product

JMS – Java Message Service

SEDA – Staged Event Driven Architecture

SOAP – Simple Object Access Protocol

JMX – Java Management Extensions

NBIO – Non-Blocking I/O

JDOM – Java-based Document Object Model

SWT – Standard Widget Toolkit

SIP – Session Initiation Protocol

CORBA – Common Object Request Broker Architecture

DCOM – Distributed Component Object Model
7. References

[1] National Software Standardization GB8567-88:


