JAWS 2: Refactorization
Framework Design and Utilization

Overview

James C. Hu
Washington University in St. Louis
jxh@cs.wustl.edu
http://www.cs.wustl.edu/~jxh/

Advisor: Dr. Douglas C. Schmidt
Sponsors: Siemens SCR
Object Technologies International
Eastman Kodak, Rochester, N.Y.
Overview

- Introduction
- JAWS 2
- Data Block
- Dispatch Policy
- IO Handler
- Protocol Pipeline Framework
- Utilization Rules
- Summary
Introduction

• Research question
  – *Identify techniques to develop Web systems that provide the best possible performance over high-speed networks.*

• Goals
  – *Alleviate “throughput preservation problem”*
  – *Create a framework to enable server developers to build high-performance Web systems by only implementing the protocol.*
A Typical Web System

- Overview of Web system components
- Targets for optimizations
JAWS 2 Overview

- Overview of JAWS top level components
- Allow different strategies to be tested independently and in concert.
JAWS 2 Overview

- Overview of JAWS inner components
- Illustration of framework use
JAWS 2: Refactorization
Framework Design and Utilization

JAWS Data Block

- Maintains server state
- Enables communication
JAWS Dispatch Policy

- Server decision maker
- Enables dynamic server behavior
**JAWS IO Handler**

- Maintains protocol and IO state
- Callback driven
Protocol Pipeline Framework

- Provides an abstraction to easily extend JAWS
- Linked list of tasks
Protocol Pipeline FURs

Framework Utilization Rules

1. Inherit from and extend JAWS_Pipeline_Handler.

2. Extract Dispatch_Policy and IO_Handler.

3. Check IO_Handler status after every IO request.

4. Return -1, 0, 1, or 2.

5. Use JAWS_Data_Block::payload() to communicate with next pipeline task.

6. If returning 2, check the payload.
Protocol Pipeline FUR #1

Inherit from and extend JAWS_Pipeline_Handler.

JAWS Pipeline Handler

- put(ACE_Message_Block *)
- handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Read

- handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Parse

- handle_put(JAWS_Data_Block *)
Protocol Pipeline FUR #2

Extract Dispatch_Policy and IO_Handler.

```
JAWS Pipeline Handler

put(ACE_Message_Block *)
handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Read

handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Parse

handle_put(JAWS_Data_Block *)
```

```
policy = db->policy ();
handler = db->handler ();
//...
```
Protocol Pipeline FUR #3

Check IO_Handler status after every IO request.

```
JAWS Pipeline Handler

put(ACE_Message_Block *)
handle_put(JAWS_Data_Block *)
```

```
JAWS HTTP 10 Read

handle_put(JAWS_Data_Block *)
```

```
JAWS HTTP 10 Parse

handle_put(JAWS_Data_Block *)
```

```
policy = db->policy ();
handler = db->handler ();
io = policy->io ();
size = db->size ();
io->read (handler,db,size);
switch (handler->status ()) {
  //...
```
Protocol Pipeline FUR #4

Return -1, 0, 1, or 2.

JAWS Pipeline Handler

put(ACE_Message_Block *)
handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Read

handle_put(JAWS_Data_Block *)

JAWS HTTP 10 Parse

handle_put(JAWS_Data_Block *)

```c
policy = db->policy ();
handler = db->handler ();
io = policy->io ();
size = db->size ();
io->read (handler,db,size);
switch (handler->status ()) {
case OK: return 0;
case IDLE: return 1;
default: return -1;
}
```
Protocol Pipeline FURs #5 and #6

Use JAWS_Data_Block::payload() to communicate with next pipeline task.
If returning 2, check the payload.
Summary

- JAWS is a framework for building flexible and adaptive Web servers.

- Enables prototyping and measuring different Web server optimization strategies.

- Previous versions of JAWS has performed as well or has outperformed top selling commercial Web servers.

- JAWS is currently being retrooled and refactored.

- Developing FURs to document how to use the framework, and perhaps automate rule adherence.