FreeBSD based Japanese Enterprise System
and
Unicage Development Method

BSD Consulting, Inc. Director / ONGS Inc. CEO / FreeBSD committer
Daichi GOTO
Summary
summary

- Profile and introduction of my FreeBSD-related works
- How about USP Lab, rapid growing enterprise system development company
- How about Unicage development method, USP’s original development method
- What I made for USP Lab
- FreeBSD based enterprise system and HPC
- A problem to be solved ASAP
Introduction
Introduction

• Daichi GOTO / 後藤大地 1980～

• FreeBSD ports / src committer unionfs, japanese ports

• BSD Consulting, Inc. Director young, new company

• ONGS Inc. CEO my own company, very small company

• Enterprise system design / development / management and maintenance, web server design / development / management and maintenance, IT-related news / articles / magazine and books writing, IT-related seminar, IT-related consulting, etc
Introduction
FreeBSD-related jobs

- FreeBSD Daily Topics (ONGS works)
  http://gihyo.jp/admin/clip/01/fdt

- FreeBSD books, magazines and articles writing (ONGS works)

- FreeBSD H/W verification services and consulting services (BSDc and ONGS works)

- FreeBSD based enterprise platform constructions and maintenance (BSDc and ONGS work) etc
Introduction
FreeBSD Digital Books

* 実践 FreeBSD サーバ構築・運用ガイド, 2012

* Practical FreeBSD server building up and management guide

ONGS works
Introduction
FreeBSD Digital Magazines


- FreeBSD Expert 2013Q2 writing is done. coming soon (maybe)

ONGS works
Introduction
FreeBSD-related activities

• FreeBSD src, ports committer
• Attend to BSDCan, EuroBSDCon, DevSummit, AsiaBSDCon, VendorSummit and writing some reports for Japanese developers and users
• FreeBSD Seminar per month
• @daichigoto tweets FreeBSD-related news and events information
Introduction
FreeBSD Daily Topics

Clang now the default on x86
2012年11月5日にFreeBSD 10-CURRENTのデフォルトコンパイラをGCCからLLVM Clangへ変更するという当初のアナウンス通り、10-CURRENTのデフォルトコンパイラがLLVM Clangへ変更されました。amd64とi386のコンパイラが次のようにかっ
cc(1), c++(1), cpp(1)はclang(1)が実体へと置き換わっています。

```
# uname -v | cut -c 1-65
FreeBSD 10.0-CURRENT #8 r242822: Fri Nov 9 21:56:12 JST 2012
# which cc c++ cpp
/usr/bin/cc
/usr/bin/c++
/usr/bin/cpp
# cc --version
FreeBSD clang version 3.2 (trunk 162187) 20120817
Target: x86_64-unknown-freebsd10.0
Thread model: posix
# c++ --version
FreeBSD clang version 3.2 (trunk 162187) 20120817
Target: x86_64-unknown-freebsd10.0
Thread model: posix
# cpp --version
FreeBSD clang version 3.2 (trunk 162187) 20120817
Target: x86_64-unknown-freebsd10.0
Thread model: posix
#
```
Japanese enterprise IT situation
Japanese enterprise IT situation
No IT sectors

• Many Japanese companies have no own IT sectors. They always outsource their system development, management and maintenance to IT vendors.

• Big companies depend on Big IT vendors.

• Middle-Small companies depend on commercial software packages.
Japanese enterprise IT situation
IT is import-dependent industry

- Japanese software industry is an import-dependent. Most softwares are not made in Japan.

- Domestic IT vendors like NEC, Hitachi and Fujitsu use imported and translated softwares from Oracle, SAP, Microsoft and so on.
Japanese enterprise IT situation
Sub-sub-sub-sub...contractor structure

- Big IT vendors play as money manager
- Sub-contractors play as project manager
- Sub-sub-contractors write specifications in excel
- Sub-sub-sub-contractors write excel documents
- Sub-sub-sub-sub-sub ... -sub...s write codes
- As a result: High costs and low efficiency
Japanese enterprise IT situation
as a result...

- Many enterprise system development projects look not working well. Too many costs, too many people, too many time, too many unnecessary documents, too many unnecessary source codes and too many stress for workers. Not happy.
Universal Shell Programming Laboratory
Universal Shell Programming Laboratory, Ltd.

- April 2005 established, Japan
- http://www.usp-lab.com/
- President is Nobuaki TOUNAKA / 當仲寛哲
- rapidly growing enterprise systems development small-middle size company
- sales accounting system, payroll accounting system, corporate system, CRM system, merchandising system, enterprise system self-manufacture etc
USP Lab
main customers

- Welcia Holdings, ANA FESTA, Ryohin Keikaku, World, Lowson, Hanshoku, Seijoishii, Yoshiduya, Tokyu Hands, Lotteria, Kitamura, Newton, Nihon Nougyo Shinbun, Trial company, Nihon Shuruihanbai, Takanagi, Sanseido, Tanakashouji etc
• They have some very unique tools to develop any enterprise systems in a day and age. The commands and a shellscript.

• They have specialized commands called “usp Tukubai” [https://uec.usp-lab.com/]

• “usp Tukubai” are 40~50 selective commands survived among from several thousands of commands they developed in past years of their businesses.

• It looks like the 40 years old UNIX-style system development.
USP Lab
their business rapidly growing

• Many business folks and developers, at first, feel disturbed and laugh at their development style to scorn

• However...

1. USP develops enterprise systems in very quick (a few days in some cases) and system works very well
2. Development cost is reasonable
3. Development is very flexible
4. Approach is very easy. At last, customer’s company could do self-manufacture (many Japanese companies have no IT sectors, they loves outsourcing)

• And, they are growing rapidly.
USP Lab

shell programming research

- NEDO (New Energy and Industrial Technology Development Organization) - Practical fast information treatment by unicage development method and pipeline calculator
- Tokyo University Tamai lab - Enterprise information system self-manufacture
- Keio University Ohiwa lab - Unicage development method and Japanese programming
- Waseda University Yamana lab - Shellscripting fast data treatment method on multicore processor
- Nagoya University Kawaguti lab - Emergency data treatment system development by Unicage development method
Unicage Development Method
Unicage Development Method

- Software Development Method for enterprise system using Unix, text file, commands and a shellscript.
- Low Cost
- Easy to Program
- Fast Development time
- Fast Processing
Unicage Development Method
key tools: texts, commands, pipelines

• Inexpensive PC is base platform
• Data is white-space separated plain text called “field-formatted text”
• Unix text processing commands (sed, awk, tr, grep, echo, cat, head, tail) and customized commands called “usptukubai”, joined by pipeline in a shellscript
Unicage Development Method
key tools: texts, commands, pipelines

Only 100 Common Commands

- cp    Copy
- find  File search
- sort  Sort
- awk   Perform operations on items
- join0 Data matching
- sm2   Sum up
- waku  Add a border
- ulock Lock control
- bdate Date management

FreeBSD Commands

- usp

Tukubai Commands

Custom Commands
Unicage Development Method
key tools: texts, commands, pipelines

Pipeline Processing

Command 1 | Command 2 | Command 3 | Command 4 | ...

Core 1
Core 2
Core 3
Core 4

Cache

CPU

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method

scalability: multicore / many-core

Pipeline Processing: easy way to use multicore

Command 1 | Command 2 | Command 3 | Command 4 | ...

Core 1
Core 2
Core 3
Core 4

Cache

CPU

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method
key tools: usp Tukubai

<table>
<thead>
<tr>
<th><strong>Database Commands</strong></th>
<th><strong>I/O Commands</strong></th>
<th><strong>Arithmetic Functions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>join0,1,2: Table join</td>
<td>cgi-name: Read data from CGI-POST</td>
<td>plus: Addition</td>
</tr>
<tr>
<td>gyo: Count matching records</td>
<td>mime-read: Read MIME encoded data</td>
<td>divsen: Divide by 1000</td>
</tr>
<tr>
<td>getfirst: Get first matching row</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getlast: Get last matching row</td>
<td></td>
<td></td>
</tr>
<tr>
<td>retu: Count columns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>sm2:</strong> Sum a field</td>
<td><strong>comma:</strong> Add commas to number</td>
<td></td>
</tr>
<tr>
<td><strong>marume:</strong> Round a number</td>
<td><strong>mojihame:</strong> Merge data into template</td>
<td></td>
</tr>
<tr>
<td><strong>ratio:</strong> Find a ratio</td>
<td><strong>tcat:</strong> Vertically concatenate</td>
<td></td>
</tr>
<tr>
<td><strong>plus:</strong> Sum all fields in a record</td>
<td><strong>ycat:</strong> Horizontally concatenate</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>map:</strong> Transpose rows/columns</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>yobi</strong> Get day of week</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>up3:</strong> Merge files on key field</td>
<td></td>
</tr>
</tbody>
</table>

*From the “Unicage Development Method Technical Overview”, 2013 USP Lab.*
• USP Labs opened license free version of usp Tukubai “Open usp Tukubai” written in Python

• I imported to FreeBSD devel/open-usp-tukubai

• http://uec.usp-lab.com/ helps you
Unicage Development Method
Fast Development / Fast Processing

• No middleware.
  Shell uses kernel’s feature (systemcalls) directory. pipe, fork, wait, open, ...

• Applications are very short (a couple dozen lines)

• usp Tukubai commands: a command has a feature, optimized for high performance.
Unicage Development Method
an application sample code 1

#!/bin/sh

join0 key=1 MASTER URE | # Join data
self 2 3 4 5 | # Select field
hsort key=1/2 | # Sort
sm2 1 2 3 4 | # Sum up
sm4 1 1 2 2 3 4 | # Intermediate total
self 1 2 4 3 | # Select Field
sm5 1 3 4 4 | # Final total
map num=1 | # Transpose
sed ‘s/A/Sales/g’ | # Text search/replace
sed ‘s/B/Profit/g’ | # Text search/replace
keta 4 6@NF-1 | # Align rows
comma 3/NF | # Add commas
cat header - | # Attach header
tocsv > result # Output to CSV
exit 0

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
#!/bin/sh

dd bs=$CONTENT_LENGTH | cgi-name > name  # Get information from web server

case "$(nameread MODE name)" in
  SEARCH)    # [Search]
    if ulock -r MST.LK; then  # Shared Lock
      nameread KEY name      |  # Get search key
        join0 key=1 – MST     |  # Search for master data
        mojihame -1LABEL html |  # Export to HTML
      fi ;;
  UPDATE)    # [Update]
    if ulock -w MST.LK; then  # Exclusive lock
      nameread -el "KEY|VAL" name > TRN.123  # Get key and value
      upl key=1 MST TRN.123 > MST.123  # Create update master
      ln -s MST.123 MST  # Allow access with same name
      cat next_html  # Output to next screen
    fi ;;
  esac

exit 0
Unicage Development Method
Flexible

• Applications are very simple and easy to learn and customize
Unicage Development Method
Data Strategy: Separate

• “To Separate is to Understand”
分ける (separate) / 分かる (understand)
The kanji “分” has 2 meanings, one is to separate, other is to understand. It’s judicious.

• Data separated by business, separated by organization, separated by software.
Unicage Development Method
Data Strategy: Distributed

- Data are copied to all software and distributed to everywhere.
- No overwrite. Applications just read a file and write into an another new file.
- Full distribution and non-overwrite system is robust for unexpected accident. Wrong data input, software bug or hardware bug. Developers can inspect easily because there are just only some text files and some little size shellscripts.
- Easy rollback
## Unicage Development Method

### Data Strategy: Distributed

<table>
<thead>
<tr>
<th>Traditional “Sharing” (centralized)</th>
<th>Full Sharing (distributed)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>One change of spec affects everyone</td>
<td>One change of spec doesn’t affect others</td>
</tr>
<tr>
<td>High Load / Program is Complex</td>
<td>Low Load / Simple Program</td>
</tr>
</tbody>
</table>

*From the “Unicage Development Method Technical Overview”, 2013 USP Lab.*
Unicage Development Method
Data Flow

Input Script
POS
Order Data
Master Record, etc.

Output Script
Screen
Report, etc.

Update/Collarate Script
Data merged
5W1H Collation
5 Layer Data Management, etc.

All three systems are created with shell scripts
Data transfer is all performed with File I/F

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method
5 Layers Data Management

Level 1
(Event Data)

Raw Data (Unsynchronized)
Data created as the information is produced
(Must not be lost; replicated and backed-up robustly)

Level 2
(Confirmed Data)

Detail Data (Synchronized)
Raw data that has been formally imported into the
system at a particular point in time

Level 3
(5W1H Data)

Level 2 data is organized and summarized in 5W1H units
(Often called “Ultra-organization.” Does not depend on
the application.

Level 4
(Application Data)

Data formatted to be easily processed by the
application

Level 5
(Output Data)

Report Image Data (Excel, PDF) or Log

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method
5 Layers Data Management

Phase 1
Level 1 (Raw Data)
Level 2
Level 3
Level 4/5 (working data)

Phase 2
Level 3
Level 4/5

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method
Development Flow

- Short Program: Fast Development
- User Focus: Efficient Communication
- 2-Steps: Accurate Spec Design

= Low Cost
Fast Response

From the "Unicage Development Method Technical Overview", 2013 USP Lab.
Unicage Development Method
Coding manners

Shell Scripts are extremely flexible so we must pay close attention to proper style when using Unicage.

- Script header style
  - Comment style
  - Variable and file naming rules
  - Rules for naming temporary files
  - One command per line
  - Transfer data using files (not environment variables)
  - Include processing is forbidden
  - File layout style
  - Execution log style
  - Rules for naming files

- Output execution start and end times
- Generate a semaphore file
- Keep it short
- Separate script and data in complex IF statements
- Delete garbage files
- Don’t create versions (but make backups)
- Multi-level calls prohibited
- Overwrite the copyright
- Understand the size of the processing file

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method Documentation

1. Very Little Documentation is Required for Development
   - Configuration of data and programs is fixed, so only basic documentation is necessary.
   - Required documents are as follows:
     - Application I/O API specifications
     - Dimensions of source data

2. Documentation for Understanding the System is Practical
   - “System Purpose”, “Business Flow”, “Manuals” are needed.
   - Most information needed to understand the system can be obtained by looking at the system operation itself (examples below)
     - Data configuration and relationships
     - Application configuration and relationships
     - Batch schedule
     - Detailed specifications (written in the shell scripts)

From the “Unicage Development Method Technical Overview”, 2013 USP Lab.
Unicage Development Method Based on the Unix Philosophy

- Small is Beautiful
- One program (command) should only do one thing
- Prototyping should be as fast as possible
- Portability takes precedence over efficiency
- Data is stored as plain text
- Commands are used as “levers” (can be combined & reused)
- Applications are written in shell script
- All programs are designed as filters (pipes)
Unicage Development Method
UEC

- http://uec.usp-lab.com/
- Web site for Unicage Engineers
- All contents are specialized for shellscript programming
- World’s most crazy shellscript site
ush / BubunFS
what I made for USP
ush

- USP’s Shell
- ash based customized shell, removed unnecessary features, added some new features including debug feature, exception handling, brace expansion and string handling
- ush is new USP’s base platform
ush

coding robustness

• ush has no unnecessary feature to improve coding quality
• ush has no features leading to some security vulnerabilities
• ush has new features to improve coding speed and reading speed
ush
error handling

# ush
err handker() {
    echo error occured
}
true
false
error occured
( true | false | true )
error occured
exit
#

# ush -e
err handker() {
    echo error occured
}

( true | false | true )
error occured
#

verbose output for debug

```bash
#!/bin/ush -exv
err handler() {
    echo error occurred
}
true
(true | false | true )
```

```bash
#!/bin/ush -exv
err handler() {
    echo error occurred
}
+5 err handler
true
+6 true
(true | false | true )
+7 true
+7 false
+7 true
+1 handler
+4 echo error occurred
error occurred
#```

```bash
# cat SAMPLE.USH
#!/bin/ush -exv
erm handler() {
    echo error occurred
}
true
(true | false | true )
#```

```bash
# ./SAMPLE.USH
#!/bin/ush -exv
err handler() {
    echo error occurred
}
+5 err handler
true
+6 true
(true | false | true )
+7 true
+7 false
+7 true
+1 handler
+4 echo error occurred
error occurred
#```
# ush

**export**

export: not found

**env**

LANG=ja_JP.UTF-8
PATH=/z/daichi/Library/bin:/z/daichi/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/z/daichi/bin

PWD=/z/dev-ush/ush/spec
TERM=xterm-256color
HOME=/z/daichi

**exit**

#
# ush
log 2> LOG.ERROR.20130518
ls /COPYRIGHT
/COPYRIGHT
ls /COPYRIGHTs
uush
false
exit
# cat LOG.ERROR.20130518
ls: /COPYRIGHTs: No such file or directory
uush: not found
#
ush
brace expansion

# ush
echo {1..5}
1 2 3 4 5
echo {1..5}{a..d}
1a 1b 1c 1d 2a 2b 2c 2d 3a 3b 3c 3d 4a 4b 4c 4d 5a 5b 5c 5d
echo FILE.{1..3}{a,d}
FILE.1a FILE.1d FILE.2a FILE.2d FILE.3a FILE.3d
exit
#
ush

substring operation

# ush
name=PRODUCT.NAME
echo $name
PRODUCT.NAME
echo ${name.1.7}
PRODUCT
echo ${name.9.11}
NAME
echo ${name.-4.4}
NAME
echo ${name.-4}
NAME
exit
#


BubunFS

- A new file which is a part of some file without data copying I/O
- kldload BubunFS
- `ln -s original "apartof seek length"`
- e.g., 1,000,000 files from a 10GB file without data read/writing I/O
BubunFS is implemented as a kernel module. You can switch on/off directory at run time.

<table>
<thead>
<tr>
<th>Id</th>
<th>Refs</th>
<th>Address</th>
<th>Size</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>0xffffffff80200000</td>
<td>150ea58</td>
<td>kernel</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0xffffffff8170f000</td>
<td>5210</td>
<td>BubunFS.ko</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0xffffffff81812000</td>
<td>390e</td>
<td>ums.ko</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0xffffffff81816000</td>
<td>12074</td>
<td>ipfw.ko</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0xffffffff81829000</td>
<td>5016</td>
<td>ipdivert.ko</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0xffffffff8182f000</td>
<td>964c</td>
<td>if_bridge.ko</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0xffffffff81839000</td>
<td>4ef3</td>
<td>bridgestp.ko</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0xffffffff8183e000</td>
<td>9c89</td>
<td>tmpfs.ko</td>
</tr>
</tbody>
</table>

```bash
go
tmpfs%
```
BOAM /home/usp/tmpfs% **ls -lh**
-rw-r--r-- 1 usp usp 4.3G May 7 13:33 data.bank

BOAM /home/usp/tmpfs% **head -5 data.bank**
0000000 20130310 101034 1000 8000
0000000 20130410 094550 2000 10000
0000000 20130430 231015 -1000 9000
0000000 20130520 042353 3000 12000
0000000 20130709 081012 4000 16000

BOAM /home/usp/tmpfs% **ln -s data.bank "a 0 34"**

BOAM /home/usp/tmpfs% **ls -l a***
lrwxr-xr-x 1 usp usp 9 May 18 22:10 a 0 34 -> data.bank

BOAM /home/usp/tmpfs% **cat a\ 0\ 34**
0000000 20130310 101034 1000 8000

BOAM /home/usp/tmpfs%
BubunFS
how to implement

- BubunFS is systemcall hock magic implementation
- BubunFS implements all related systemcalls to put the BubunFS feature into place. BubunFS kernel module replaces some FreeBSD’s default systemcalls with BubunFS’s systemcalls at runtime.
- We choose to use symbolic link file as a trick of BubunFS because it’s the most fastest one.
BubunFS
use case

• One big Master file (10GB)

• Some applications want to use a part of the MASTER file. e.x. 100 applications read the Master file | grep > 100-new-small-files ... very slow

• BubunFS can create some million small files in seconds
ush / BubunFS

next step

• ush2 - more debugging features, remote control, network programming

• GattaiFS - reverse feature of BubunFS. A file consisted by any other files.
uspBOA
usp BigData Oriented Architecture

- "usp BOA" FreeBSD based BigData processing appliance (PC based cluster) for USP Lab
- 1 master, 5 slaves PCs. 10GbE NIC connected network
- 1 billion records sort: 90sec, great cost-benefit
- 1 billion records search: 4sec, great cost-benefit
**uspBOA**

**uspBOA Architecture**

**Master Server**
- CPU: Xeon E5-2687W 3.10GHz x2
- Memory: 128GB
- HDD: SATA 1TB

10Gbps Network

**Slave Servers**
- 5 PCs
- Store massive data
- Perform actual processing

**Shell Script is written only on this server**

**Optimum data structure is calculated**

**Can scale out easily as the amount of data grows…**

From the “Big Data...Small pricetag”, 2013 USP Lab.
uspBOA
1st BOA : failed

• 1st BOA cluster - I chose Mellanox Technologies InfiniBand ConnectX-II for network with OFED

• That works. Great

• But unstable. Useless

• We have no time to change kernel source code. So we chose to buy other devices
usp BOA

2nd BOA : not enough

- 2nd BOA cluster - I chose Intel X540-T2, and works well
- Good
- But we need more processors power, more impact.
uspBOA
3rd BOA : success

• 3rd BOA cluster - we replaced all master and slaves CPU to 6core/12thread Core i7-3960X 3.30GHz and Xeon E5-2687W 3.10GHz.

• Good. Enough impact
USPBOA

USPBOA (3rd) Architecture

Master Server

- CPU: Xeon E5-2687W 3.10GHz x2
- Memory: 128GB
- HDD: SATA 1TB

Slave Servers

- 5 PCs
- Store massive data
- Perform actual processing

- CPU: Core i7-3960X 3.30GHz
- Memory: 32GB
- HDD: SATA 1TB

Optimum data structure is calculated

Can scale out easily as the amount of data grows...

From the “Big Data...Small pricetag”, 2013 USP Lab.
#!/bin/ush -x
#Example script for summing a large data set

clust-join1 slavefile1 key=1 master URE   |   # Data JOIN (Bigdata)
para-self 10 2/NF-1                       |   # SELECT(Bigdata)
clust-sm2 slavefile2 1 2 3 4              |   # SUM
sm4 1 1 2 2 3 4                           |   # Intermediate sum
self 1 2 4 3 5                            |   # SELECT
sm5 1 3 4 4                               |   # TOTAL
map num=1                                 |   # Transform
sed 's/A/Sales/g'                         |   # Text replace
sed 's/B/Profit/g'                        |   # Format columns
keta 4 6@NF-1                             |   # Insert commas
comma 4 5                                 |   # Output with header
cat header -> result                      |
exit 0

“slavefile” contains the names of the slave servers and the number of parallel processes
<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select (para-grep)</td>
<td>Select all records starting with the text “123” from among 1 billion records using 10 parallel processes</td>
<td>3 secs.</td>
</tr>
<tr>
<td>2. Sort (clust-qsort)</td>
<td>Sort 1 billion random records in ascending order using 40 parallel processes on 5 slave servers</td>
<td>97 secs.</td>
</tr>
<tr>
<td>3. Sum (clust-sm2)</td>
<td>Sum key fields in 1 billion random records using 40 parallel processes on 5 slave servers</td>
<td>35 secs.</td>
</tr>
<tr>
<td>4. Mathematical Operations</td>
<td>Perform mathematical calculations between fields on 1 billion records using 40 parallel processes on 5 slave servers</td>
<td>22 secs.</td>
</tr>
<tr>
<td>(clust-lcalc)</td>
<td>Perform precision floating-point operations</td>
<td>67 secs.</td>
</tr>
<tr>
<td>5. Join (clust-join1)</td>
<td>Perform a join operation on 1 billion records using 40 parallel processes on 5 slave servers. The master server is relatively small.</td>
<td>37 secs.</td>
</tr>
<tr>
<td>6. Complicated Operations</td>
<td>Distribute 1 billion records by key block units and perform several calculations (key sumup, average, round, literal) using 40 parallel processes on 5 slave servers</td>
<td>17 secs.</td>
</tr>
</tbody>
</table>

From the “Big Data Software Appliance Simple, High-Speed Big Data Processing using Shell Scripting and uspTukubai”, 2013 USP Lab.
# uspBOA Benchmarks

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Big Data Select (apli-select)</td>
<td>Perform a matching select on 10,000 transactions (join and exclude) from among 10 billion records distributed across the slave servers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.5 secs.</td>
</tr>
</tbody>
</table>
| 2. Big Data Update (Add & Change, Delete, Sum) | 2. Big Data Update (Add & Change, Delete, Sum)  
apli-update  
apli-delete  
apli-sumup                                                                                                                  | 5.5 secs. |
| 3. Big Data Search (apli-search) | Search account holder data based on Rank, Gender, Geographical Region, Age Group, Length of Membership and Minimum Average Score from among 10 billion records distributed across the slave servers                  | 1.2 secs. |

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
## Examples

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>OLD: COBOL Program on Large Server (15hrs. 29mins)</th>
<th>NEW: UNICAGE Program on 5 PCs (1hr. 56mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Batch Processing</strong> (Leading Credit Card Company)</td>
<td>Processing of daily transaction details on 60,000,000 credit card accounts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OLD: COBOL Program running on Large Server (15hrs. 29mins)</td>
<td>NEW: UNICAGE Program running on 5 PCs (1hr. 56mins)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEW: UNICAGE Program running on 5 PCs (1hr. 56mins)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><strong>Complex ETL</strong> (Leading Investment Bank)</td>
<td>Data Creation for DB Loading of 30,000,000 daily transaction records</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OLD: JAVA + PostgreSQL (90 minutes)</td>
<td>NEW: Unicage Program running on 1 PC (91.58 seconds)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEW: Unicage Program running on 1 PC (91.58 seconds)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><strong>Complex ETL</strong> (Large Electric Utility)</td>
<td>Preprocessing of 10GB of Smart Meter data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OLD: JAVA on HPUX Itanium 1.6GHz/2Core (15 hours)</td>
<td>NEW: Unicage Program running on 1 PC (FreeBSD 9.1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEW: Unicage Program running on 1 PC (FreeBSD 9.1) (4 mins 16 secs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>Large Data Search</strong> (Biggest Search Engine in Korea)</td>
<td>50.3 Billion Log Records from 5 years (19.2TB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 Types of SQL Searches translated to Unicage</td>
<td>Search Time: 0.227 sec - 4.763 sec</td>
<td></td>
</tr>
</tbody>
</table>

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
Bigdata case studies using Unicage

(1) Replacement of Batch Processing System  
   (Major Credit Card Company)

(2) Complex ETL  (Investment Bank)

(3) Complex ETL  (Electric Power Utility)

(4) Search of Large Data Set  (Korean Search Engine)

From the “Big Data...Small pricetag”, 2013 USP Lab.
Large data set is processed on the host. This processing will be ported to Unicage. We receive the data that needs processing from the host, Unicage performs some processing, then compare.

**Diagram Description:**
- Databases are connected to Large Dataset Processing (1).
- Large Dataset Processing (1) connects to Upload Processing.
- Upload Processing connects to Flat File.
- Flat File connects to Receive in Unicage Server.
- Unicage Server connects to Large Dataset Processing.
- Large Dataset Processing connects to Compare.
- Compare connects to Large Dataset Processing (30) and Large Dataset Processing (50).

From the “Big Data...Small pricetag”, 2013 USP Lab.
### uspBOA

#### Processing Speed

- Processing time was reduced to 1/8 of the COBOL system
  \( \frac{116.00}{929.69} = 12.4\% \)
- Unicage was measured running on 5 x86 servers (6-core CPU x 2, 48GB RAM)
- If the number of servers is increased and processing is distributed, even faster processing is possible.

<table>
<thead>
<tr>
<th></th>
<th>COBOL</th>
<th>Unicage (Single x86 Server)</th>
<th>Unicage (Five x86 Servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time</td>
<td>929.69 mins. (15 hrs. 29 mins.)</td>
<td>313.58 mins. (5 hrs. 13 mins.)</td>
<td>116.00 mins. (1 hr. 56 mins.)</td>
</tr>
<tr>
<td>Hardware</td>
<td>Host</td>
<td>Single x86 Server</td>
<td>Five x86 Servers</td>
</tr>
<tr>
<td></td>
<td>• Initial Investment over $1M</td>
<td>• Dual 6-core CPUs</td>
<td>• Dual 6-core CPUs</td>
</tr>
<tr>
<td></td>
<td>• Maintenance Fee also High</td>
<td>• 48GB RAM</td>
<td>• 48GB RAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 x HDD (SATA 2TB)</td>
<td>• 2 x HDD (SATA 2TB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Initial Investment $10K</td>
<td>• Initial Investment $50K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Maintenance Fee is Low</td>
<td>• Maintenance Fee is Low</td>
</tr>
</tbody>
</table>

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
## Development Productivity

Using COBOL
24 processes and 7 jobs required, so development took 3 months.

Using Unicage
- Coding: 5 days
- Testing: 5 days
- Performance Tweaking: 3 days

- Developed by a Unicage engineer with 5 years experience in 13 days.

<table>
<thead>
<tr>
<th></th>
<th>COBOL</th>
<th>Unicage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Processes</td>
<td>7 Jobs &amp; 24 Processes</td>
<td>11 Shell Scripts</td>
</tr>
<tr>
<td>Development Time</td>
<td>3 Months</td>
<td>13 days</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>3,645</td>
<td>981</td>
</tr>
</tbody>
</table>

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
(2) Complex ETL (Investment Bank)

• Using the Unicage development method, we will perform reformatting of data so that it is in a format that can be loaded into the transaction storage database.
• We will then compare processing time.

Transaction Log

<table>
<thead>
<tr>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child 1</td>
</tr>
<tr>
<td>Grandchild 1-1</td>
</tr>
<tr>
<td>Grandchild 1-2</td>
</tr>
<tr>
<td>Child 2</td>
</tr>
<tr>
<td>Grandchild 2-1</td>
</tr>
<tr>
<td>Parent</td>
</tr>
<tr>
<td>Child 1</td>
</tr>
<tr>
<td>Child 2</td>
</tr>
<tr>
<td>Parent</td>
</tr>
<tr>
<td>Child 1</td>
</tr>
<tr>
<td>Child 2</td>
</tr>
</tbody>
</table>

Hierarchical Multi-Layout

Record Types (approx. 100)

<table>
<thead>
<tr>
<th>A</th>
<th>Parent</th>
<th>Child1</th>
<th>Grandchild1–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Parent</td>
<td>Child1</td>
<td>Grandchild1–2</td>
</tr>
<tr>
<td>A</td>
<td>Parent</td>
<td>Child1</td>
<td>Grandchild2–1</td>
</tr>
<tr>
<td>B</td>
<td>Parent</td>
<td>Child1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Parent</td>
<td>Child2</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Parent</td>
<td>Child1</td>
<td>Child2</td>
</tr>
</tbody>
</table>

Data to be Loaded

Layout resolves the Parent/Child/Grandchild relationships

Execution Speed using Java+ PostgreSQL is about 90 minutes

From the “Big Data...Small pricetag”, 2013 USP Lab.
## uspBOA Processing Speed

### Development/Testing Environment

| Computer                | Desktop PC  
|-------------------------|-------------
|                         | (Intel Core i7 processor, 16GB RAM) |
| Operating System        | FreeBSD 9.0 Release#0 |
| Shell Commands          | USP Unicage Enterprise Version |

<table>
<thead>
<tr>
<th>Application</th>
<th>Details</th>
<th>Records Processed</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS–MASTER</td>
<td>Top Shell</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>PROCESS–001</td>
<td>Exception Processing 1</td>
<td>8,327</td>
<td>8</td>
</tr>
<tr>
<td>PROCESS–002</td>
<td>Exception Processing 2</td>
<td>117,838</td>
<td>9</td>
</tr>
<tr>
<td>PROCESS–003</td>
<td>Exception Processing 3</td>
<td>81</td>
<td>11</td>
</tr>
<tr>
<td>PROCESS–004</td>
<td>Exception Processing 4</td>
<td>5,028</td>
<td>19</td>
</tr>
<tr>
<td>PROCESS–005</td>
<td>Exception Processing 5</td>
<td>332</td>
<td>14</td>
</tr>
<tr>
<td>PROCESS–006</td>
<td>Normal Processing</td>
<td>27,614,260</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29,015,393 (4.36 GB)</td>
<td>96</td>
</tr>
</tbody>
</table>

**Execution Speed:**

- Real: 91.58 sec
- User: 132.85 sec
- Sys: 22.53 sec

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
The legacy system converts the character set from native to SJIS. We ported this process to Unicage. We confirmed the input and output files are the same and calculated the difference in processing speed using Unicage.
**uspBOA**

**Processing Speed**

We tested on 2GB, 5GB and 10GB data sets.

We used the following server environment:

- **Java:** HP-UX, Itanium 1.60GHz 2core, 4GB
- **Unicage:** FreeBSD, Core i7 4core, 16GB, SATA (2TB)

<table>
<thead>
<tr>
<th>Data Amount</th>
<th>2GB</th>
<th>5GB</th>
<th>10GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,240,555 records</td>
<td>3hrs 7mins 53secs</td>
<td>7hrs 30mins</td>
<td>15 hrs</td>
</tr>
<tr>
<td>18,095,303 records</td>
<td>43.411secs</td>
<td>1 min 49.085secs</td>
<td>4mins 16.906secs</td>
</tr>
<tr>
<td>36,178,437 records</td>
<td>11273/43.411=259x faster</td>
<td>27000/109.085=247x faster</td>
<td>54000/256.906=210x faster</td>
</tr>
</tbody>
</table>

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
(4) Search of Large Data Set (Korean Search Engine)

Analysis of search logs from a major search engine site
Analysis based on text search and user IP address search

Expected data: 10.8GB/day x 365 days x 5 years = 19.2TB
(27,610,000 records)                     (50 Billion)

Shell Script + Pompa

WebServer (distribution)

Front-end Terminals

Unicage Server Cluster
0.5TB x 40 servers

From the “Big Data...Small pricetag”, 2013 USP Lab.
B3: Count number of records for each C_QUERY_NOSP, C_USER
B4: Count number of records for each C_USER, output counts over 30
B5: Output C_QUERY_NOSP list using conditions C_DATE and C_USER
B6: Count number of records for each C_REQ_FRM, output row counts in descending order
B7: Count number of records for each C_CONNECTION
B8: Count number of records for each C_QUERY_NOSP using conditions C_DATE and C_CONNECTION
B9: Count number of records for each C_QUERY_NOSP with C_CONNECTION‘X’ over 500
B10: Count number of records for each C_QUERY_NOSP with unique C_SESSION1 over 3
B11: Count number of records for each C_QUERY_NOSP that don’t occur on a specific date
B12: Count number of records with C_IP of 3 or higher and count number of records with unique C_QUERY_NOSP
**uspBOA**

**SQL and Shell Programming**

Shows equivalent shell script for each SQL code

<table>
<thead>
<tr>
<th><strong>B3 [SQL]</strong></th>
<th><strong>B3 [USP]</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select C_QUERY_NOSP, C_USER, count(*) from SEARCHLOG where C_DATE='2006-09-18'</code></td>
<td>`cat ${lv3d}/L3.DAY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>B9 [SQL]</strong></th>
<th><strong>B9 [USP]</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select A.q1, A.cnt1 as a1, B.cnt2 as a2 from (select C_QUERY_NOSP as q1, count(*) as cnt1 from searchlog where C_DATE='2006-09-18' and C_CONNECTION='X' group by C_QUERY_NOSP having</code></td>
<td>`cat ${lv3d}/L3.DAY</td>
</tr>
</tbody>
</table>

*From the “Big Data...Small pricetag”, 2013 USP Lab.*
BSDc for Enterprise
BSDc for Enterprise
BSD Consulting, Inc.

- Established 1st June, 2012
- wholly owned subsidiary of USP Lab.
- short name, BSDc
- President: Nobuaki TOUNAKA / 當仲寛哲
  Director: Daichi GOTO / 後藤大地
BSDc for Enterprise
2 years ago

- President Tounaka have involved me as a FreeBSD consultant 2 years ago.
- USP found that FreeBSD is better choice for them as base platform. Until this time, they used CentOS and bash. I push them FreeBSD and ash.
- I developed the customized shell (ush) and specialized filesystem for their business (BubunFS).
BSDc for Enterprise

customers needed us

• a certain USP’s customer hesitated to take FreeBSD as their base platform.

• They said, because of the lack of the company for support of FreeBSD, they could not choose FreeBSD.

• Exactly, we lacked FreeBSD support company.

• So, we established “BSD Consulting, Inc.” for our business.
• FreeBSD Supporting and Consulting services
• Providing FreeBSD Information in Japanese
• FreeBSD H/W verification service
• FreeBSD Seminar services
BSDc for Enterprise
FreeBSD information in Japanese

• Most japanese can not understand English.
• My English is lesser, but others are terrible.
• Folks attended AsiaBSDCon 2013 already know that, uh?
• Release note, Errata, Security Advisory in Japanese are valuable contents.
BSDc for Enterprise
H/W verification

- Japanese domestic server H/W vendors lack of FreeBSD support, because of the lack of FreeBSD support company

- If H/W vendors say “our products work with FreeBSD 9.1-RELEASE”, that’s good for all FreeBSD users and customers
BSDc for Enterprise
NEC Express5800 Verification

• They changed their on-board NIC chipset from Intel to Broadcom because of the cost a year ago

• They choose the new MegaRAID card that does not work with mfi

• They needed some patches
FreeBSD didn’t work on NEC’s new Express5800. Their customer got angry.

BSDc and NEC have a contract about FreeBSD support and H/W verification.

Some Express5800 series will work with FreeBSD.

Patches, documents and information will be open on BSDc website.
BSDc for Enterprise
NPO for enterprise

• We started to establish two organizations at June 2012. One is BSDc, other is NPO for *BSD “BSD Research (BSDr)”

• chairman : Sato-san (aka hrs)

• will be established at Summer, 2013

• core business : AsiaBSDCon, BSD Certification, *BSD documents translation
BSDc for Enterprise
BSDc and BSDr

• We thought that we need a fair and impartial certified organization to promote FreeBSD to enterprises company

• BSD Certification / BSD Certification Group is qualify

• NPO cooperates with BSD CG, and provides BSD Certification in Japanese
BSDc for Enterprise
translation ongoing

- Japanese Documents are critical for all Japanese FreeBSD users
- NPO for *BSD are trying to translate important FreeBSD relative documents into Japanese
A problem to be solved ASAP
A problem to be solved ASAP
InfiniBand driver and OFED

- HPC needs InfiniBand driver. 10GbE works fine. But InfiniBand transports 3.2 times faster than 10GbE.

- In fact, we are constructing new uspBOA with Linux, because of InfiniBand.
A problem to be solved ASAP
InfiniBand driver and OFED

- We have tried to improve OFED on FreeBSD last 2~3 months
- In the end, it failed. At last we realized that we were implementing all Linux NAPI in the FreeBSD kernel. It looks like a wrong approach.
- We are considering next approach. If you have any ideas, please contact me.
A problem to be solved ASAP
InfiniBand driver and OFED

• We need InfiniBand drivers. The lack of InfiniBand drivers give RadHat/CentOS some advantage as common HPC platform.

• Should we contact to Mellanox Technologies?

• Should we suggest to FreeBSD Foundation to develop latest OFED subsystem?

• Current big concern. Big business showstopper
Question?