
# Unicage Development Method
# Big Data Case Studies

**Universal Shell Programming Laboratory, Ltd.**

**February 2013**

Universal Shell Programming Laboratory

# Big Data Case Studies using Unicage

usp lab.

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)***

⑤ **Summary**

# ① Replacement of Batch Processing System

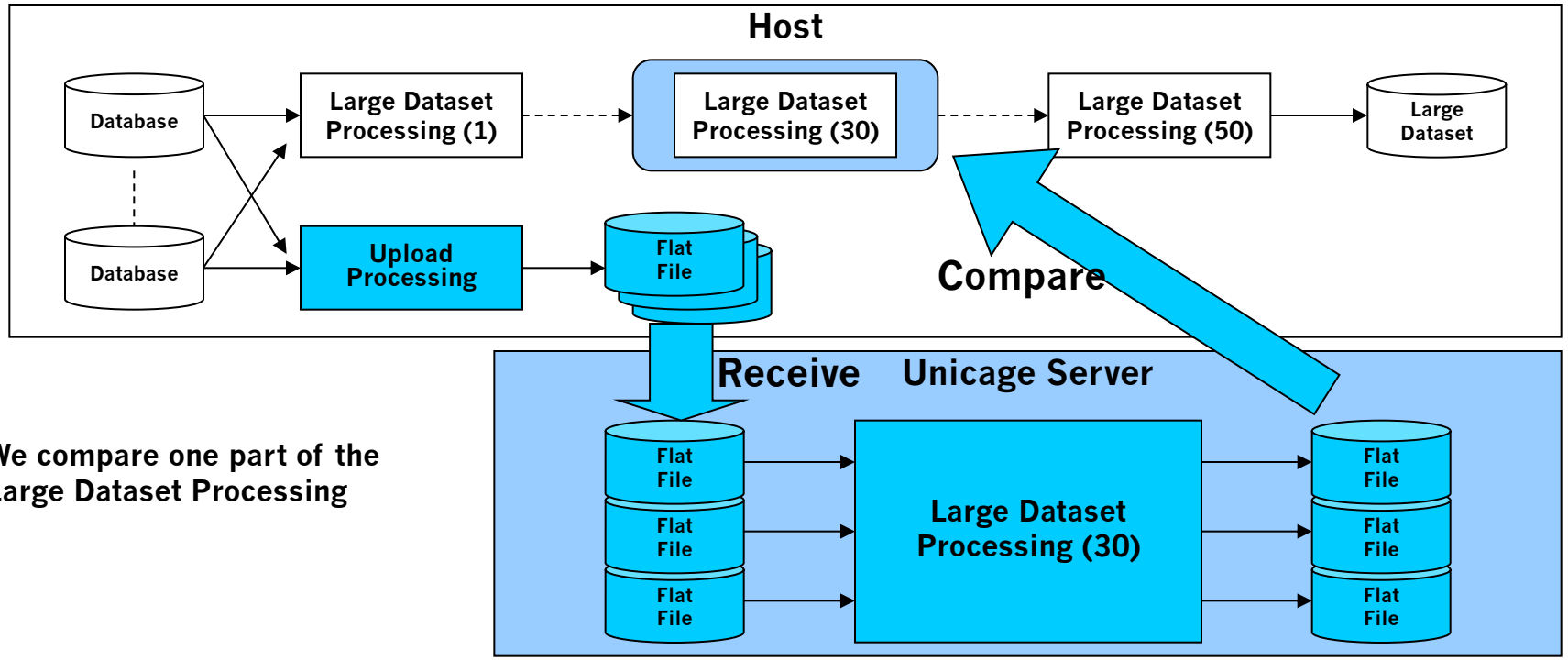
*(Major Credit Card Company)*

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.**




**We compare one part of the Large Dataset Processing**

Universal Shell Programming Laboratory

# Processing Speed

usp lab.

- Processing time was reduced to 1/8 of the COBOL system (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.

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

# Development Productivity

usp lab.

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

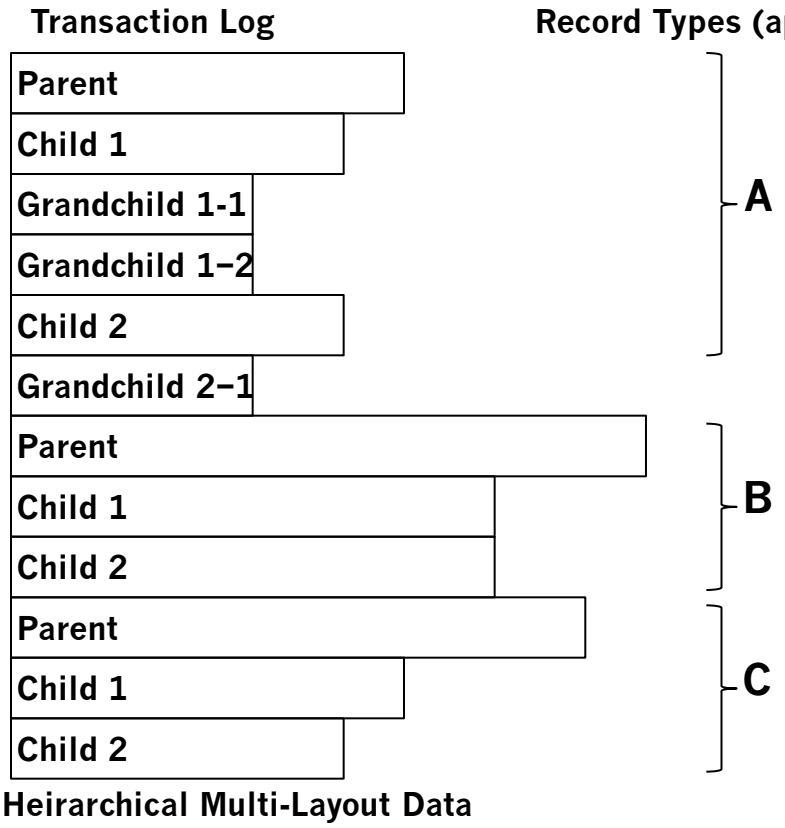
- **Using Unicage**
  **Coding: 5 days**
  **Testing: 5 days**
  **Performance Tweaking: 3 days**
- **Developed by a Unicage engineer with 5 years experience in 13 days.**

| | COBOL | Unicage |
|---|---|---|
| Number of Processes | 7 Jobs & 24 Processes | 11 Shell Scripts |
| Development Time | 3 Months | 13 days |
| Lines of Code | 3,645 | 981 |

# ② Complex ETL *(Investment Bank)*

usp lab.

- **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.**




**Data to be Loaded in DB**

| | | | |
|---|---|---|---|
| A | Parent | Child1 | Grandchild1−1 |
| A | Parent | Child1 | Grandchild1−2 |
| A | Parent | Child1 | Grandchild2−1 |
| B | Parent | Child1 | |
| B | Parent | Child2 | |
| C | Parent | Child1 | Child2 |

**Layout resolves the Parent/Child/Grandchild relationships**

Execution Speed using Java+ PostgresSQL is about 90 minutes

Universal Shell Programming Laboratory

# Processing Speed

usp lab.

Development/Testing Environment

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

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

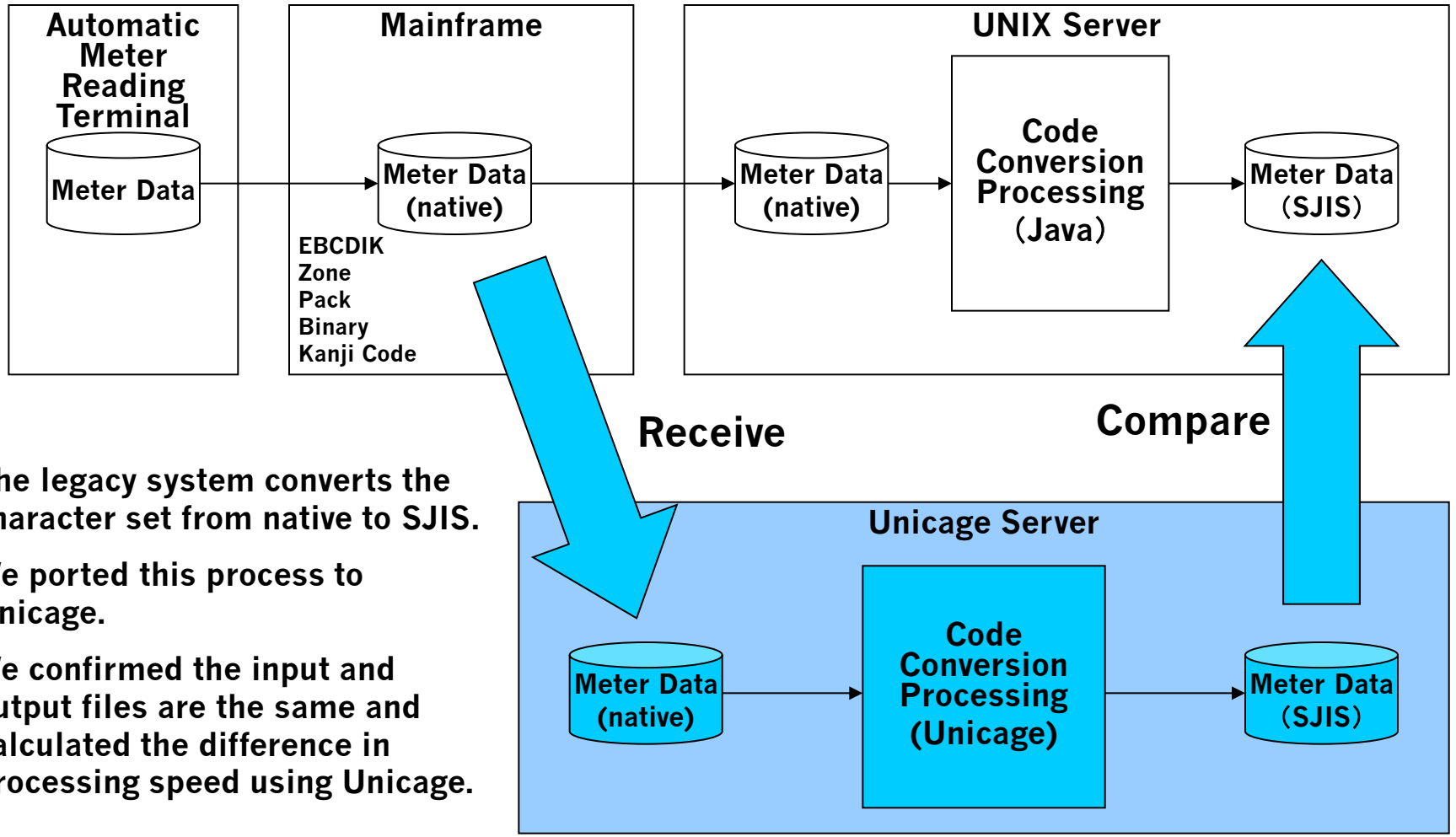
**Execution Speed:**

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

# ③ Complex ETL (Electric Power Utility)

usp lab.

## Character set conversion of host data (from native to SJIS)




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.

Universal Shell Programming Laboratory

# Processing Speed

usp lab.

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)

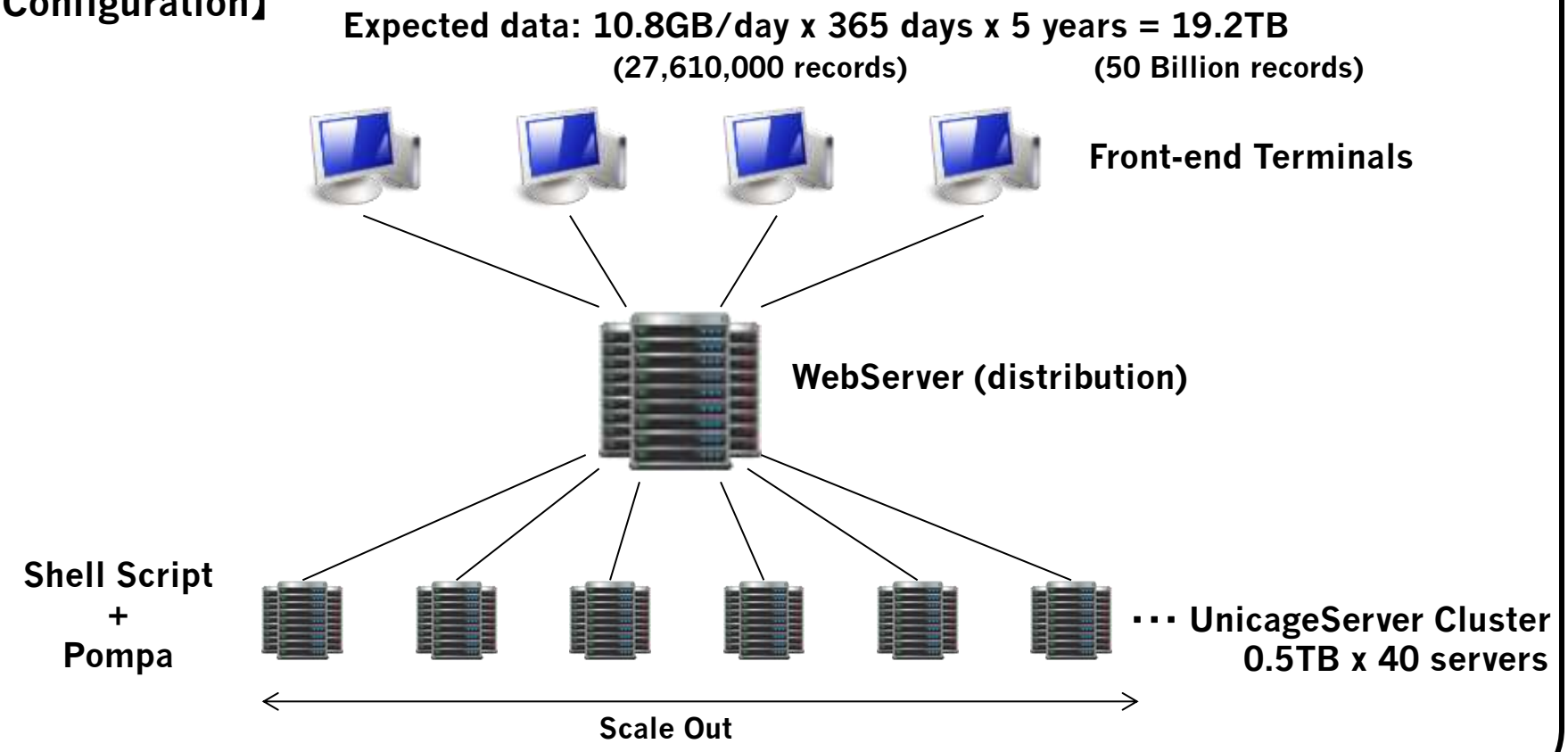
| Data Amount | 2GB 7,240,555 records | 5GB 18,095,303 records | 10GB 36,178,437 records |
|---|---|---|---|
| Java | 3hrs 7mins 53secs | 7hrs 30mins | 15 hrs |
| Unicage | 43.411secs | 1 min 49.085secs | 4mins 16.906secs |
| Difference | 11273/43.411= 259x faster | 27000/109.085= 247x faster | 54000/256.906= 210x faster |

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

usp lab.

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

**【Configuration】**




Universal Shell Programming Laboratory

# SQL and Shell Programming (1/2)


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

# SQL and Shell Programming (2/2)

usp lab.

2. Shell Programming Example

Shows equivalent shell script for each SQL code

```
B3【SQL】:
select C_QUERY_NOSP, C_USER, count(*)
 from SEARCHLOG
 where C_DATE='2006-09-18'
 group by C_QUERY_NOSP, C_USER;
```

```
B3【USP】:
cat ${lv3d}/L3.DAY     |
awk '$4=="20060918"    |
self 23 16             |
dsort key=1/2          |
count 1 2
```

```
B9【SQL】:
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 count(*)>500 ) A,
(select C_QUERY_NOSP as q2, count(*) as cnt2
  from searchlog
  where C_DATE='2006-09-18' and C_CONNECTION<>'X'
  group by C_QUERY_NOSP) B
 where A.q1=B.q2 order by a1 desc, a2 asc;
```

```
B9【USP】:
cat ${lv3d}/L3.DAY                  |
awk '$4=="20060918"&&$14!="X"'      |
self 23                             |
dsort key=1                         |
count 1 1                           > $tmp-b
cat ${lv3d}/L3.DAY                  |
awk '$4=="2006-09-18"&&$14=="X"'    |
self 23                             |
dsort key=1                         |
count 1 1                           |
awk '$2>500'                        |
join1 key=1 $tmp-b -                |
sort -k2,2nr -k3,3n
```

# Processing Speed

usp lab.

## Development/Testing Environment

| | |
|---|---|
| Computer | Desktop PC<br>(Intel Core i7 processor, 16GB RAM) |
| Operating System | FreeBSD 9.0 Release#0 |
| Storage | SATA HDD (1) |
| Shell Commands | USP Unicage Enterprise Edition |

| | Corresponding SQL | Execution Time (MIN) | Execution Time (MAX) | Execution Time (AVG) |
|---|---|---|---|---|
| #01 | B3 | 1.132 | 1.357 | 1.235 |
| #02 | B4 | 0.139 | 0.140 | 0.139 |
| #03 | B5 | 0.002 | 0.003 | 0.002 |
| #04 | B6 | 0.002 | 0.002 | 0.002 |
| #05 | B7 | 1.154 | 1.155 | 1.154 |
| #06 | B8 | 0.030 | 0.030 | 0.030 |
| #07 | B9 | 2.673 | 2.898 | 2.748 |
| #08 | B10 | 1.440 | 1.440 | 1.440 |
| #09 | B11 | 4.760 | 4.766 | 4.763 |
| #10 | B12 | 0.006 | 0.006 | 0.006 |

# ⑤ Summary

usp lab.

**Challenge #1【Reduced Performance】**

**As the amount of data increases and as the business logic changes repeatedly, processing performance gradually decreases, causing problems for the business.**

**Challenge #2【Cost】**

**Requires specialized high-performance hardware and advanced middleware, increasing the initial investment cost and ongoing maintenance cost.**

**【Legacy Methods】**

- **Purchase and deploy the latest high-performance specialized hardware and advanced middleware.**
  - **Performance is improved, but costs skyrocket.**
- **Re-write software using latest techniques (Hadoop, etc.)**
  - **High Cost. Difficult to recruit and train engineers.**

**【Background】**

**As the precision of data and the storage of past data increases, the amount of data increases to the point that legacy Relational Databases cannot handle.**

# Why is Unicage Fast? (1/2)

usp lab.

## ①We do not use middleware with huge overhead

We use only the core functions of the OS, without any database, runtime or middleware. From this aspect, UNIX/Linux OSes like FreeBSD are excellent since they have compact kernel code and you can select the required peripheral software from the PORTS collection.

## ②USP Unicage commands have been precisely tuned

We have developed the commands used in the shell scripts in the C language and they control memory and CPU directly. They are extensively tuned, for example by using the SIMD command inline. For this reason, it is tens of times faster than commands written in Java. (This is clear by the difference in the size of the post-compilation assembler code.)

## ③Parallel Processing using Pipelines

Shell scripts can easily use the ‘‘pipe’’ which is a unique feature of UNIX. By connecting USP Unicage commands with a pipeline you can achieve parallel processing which improves processing speed. In one project for an investment bank, we utilized 95% of CPU in a 16-core machine to process 30 million records 60 times faster than their legacy system.

# Why is Unicage Fast? (2/2)

usp lab.

## ④ ush

**In order to eliminate the overhead of the shell itself, we have created our own shell called ‘‘ush’’ which is based on ‘‘ash’’. The same shell script runs 1.7 times faster on ‘‘ush’’ than on standard ‘‘bash’’. We continue to improve the ‘‘ush’’ shell, for example by changing the implementation of pipes to ‘‘mmap’’ (kernel memory) with ID passing.**

## ⑤ Pompa Technology

**In order to search large datasets, we employ directory tree division and memory cache control. Our ‘‘Pompa Technology’’ embeds the search key in the path name, enabling two-layer search at the OS level and the Unicage level. Using this technology we were able to return search results from 10TB of log data (from a Korean search engine) in less than 0.1 second without using expensive appliances.**