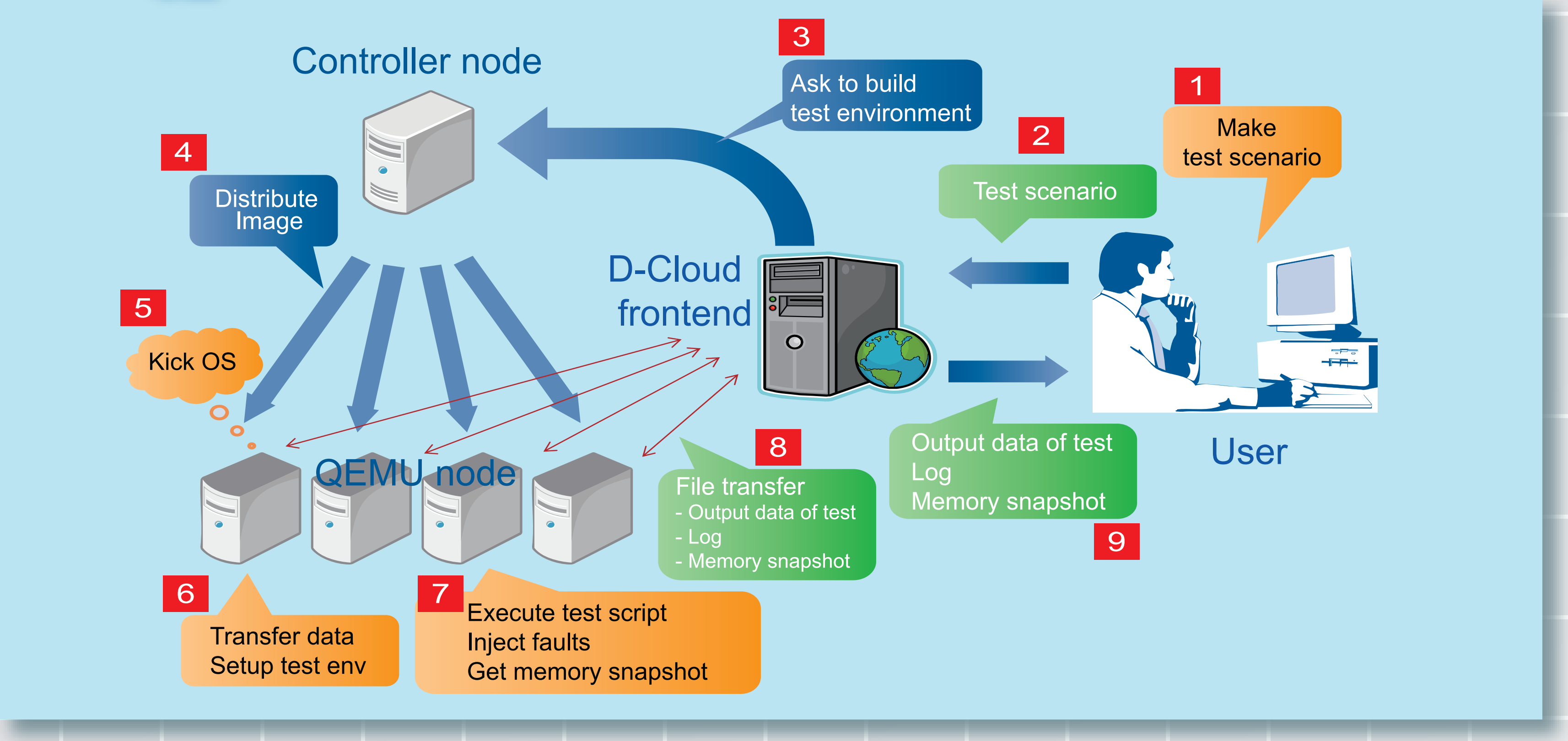
Center for Computational Sciences, University of Tsukuba
www.ccs.tsukuba.ac.jp

High Performance Computing Research

D-Cloud: Large-scale Test Farm using Cloud-computing System

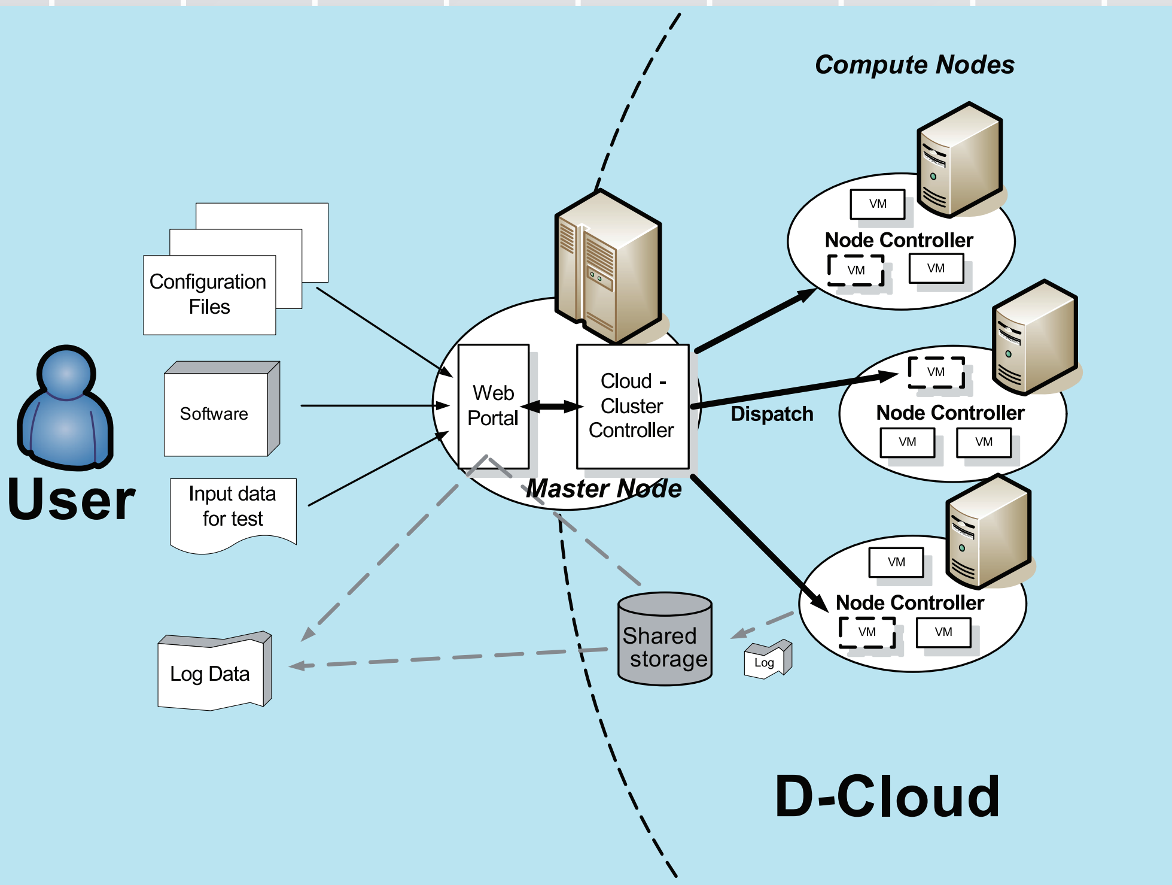
- Background & Objective
  - In order to reduce potential factors causing failure, software components should be tested carefully and exhaustively
  - There are many demands for environments to perform many tests rapidly
- D-Cloud is ...
  - An environment to helps process to improve software dependabilitye
  - To accelerate testing process through parallel test execution utilizing large computation resource managed by Cloud computing system
    - ► Eucalyptus（like Amazon EC2, Open-source）is used
  - VM fault-injection facility(Fault VM/QEMU) is available for testing HA software

Testing procedure



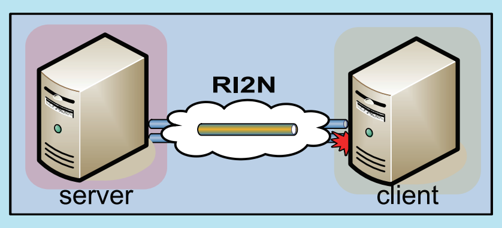
Architecture

- D-Cloud consists of multiple compute nodes which executes tests and the master node which manages them
- The master node deploys VM instances on compute nodes on demand
- Users access D-cloud through web portal offered by the master node



Configuration file for testing

- D-Cloud executes a series of tests following a configuration file
- An example of a configuration file is shown below
  - In this example, a fault is injected 100sec after booting, then the test is halted 200sec after the injection



RI2N : multi-link fault-torerant NW developed on HPCS lab

Machine Definition

```
<machineDefinition>
    <machine>
        <name>server</name>
        <cpu>1</cpu>
        <mem>500000</mem>
        <nic>3</nic>
        <id>emi-1F8A1210</id>
    </machine>
    <machine>
        <name>client</name>
        <cpu>1</cpu>
        <mem>500000</mem>
        <nic>3</nic>
        <id>emi-178911E1</id>
    </machine>
</machineDefinition>
```

System Definition

```
<systemDefinition>
    <systemconf>
        <name>systemA</name>
        <host>
            <hostname>server</hostname>
            <machinename>server</machinename>
            <config>servconf</config>
        </host>
        <host>
            <hostname>client</hostname>
            <machinename>client</machinename>
            <config>clientconf</config>
        </host>
    </systemconf>
</systemDefinition>
```

Injection Def & Scenario

```
<injectionDefinition>
    <injection>
        <name>injectionA</name>
        <fault>
            <location>network</location>
            <target>eth0</target>
            <kind>loss</kind>
            <time>20</time>
        </fault>
    </injection>
</injectionDefinition>
<testDescription>
    <run>
        <name>testA</name>
        <systemname>systemA</systemname>
        <halt when="300">down</halt>
        <script>
            <on>client1</on>
            <putFile>test.sh</putFile>
            <exec>test.sh</exec>
            <inject when="100">injectionA</inject>
        </script>
    </run>
</testDescription>
```

FFTE: A High-Performance FFT Library

- FFTE is a Fortran subroutine library for computing the Fast Fourier Transform (FFT) in one or more dimensions.
- It includes complex, mixed-radix and parallel transforms.
- FFTE is typically faster than other publically-available FFT implementations, and is even competitive with vendor-tuned libraries.

Features

HPC Challenge benchmark

- High speed
  - Supports Intel's SSE2/SSE3 instructions.
- Parallel transforms
  - Shared / Distributed memory parallel computers (OpenMP, MPI and OpenMP + MPI)
- High portability
  - Fortran77 + OpenMP + MPI
  - Intel's intrinsics for SSE2/SSE3 instructions.
- HPC Challenge Benchmark
  - FFTE's 1-D parallel FFT routine has been incorporated into the HPC Challenge (HPCC) benchmark.

Design

- Performance
  - One goal for large FFTs is to minimize the number of cache misses.
- Ease of use: routine interfaces
  - Similar to sequential SGI SCSL or Intel MKL routines
- Portability
  - Communication: MPI
  - Computation: Fortran77 + OpenMP

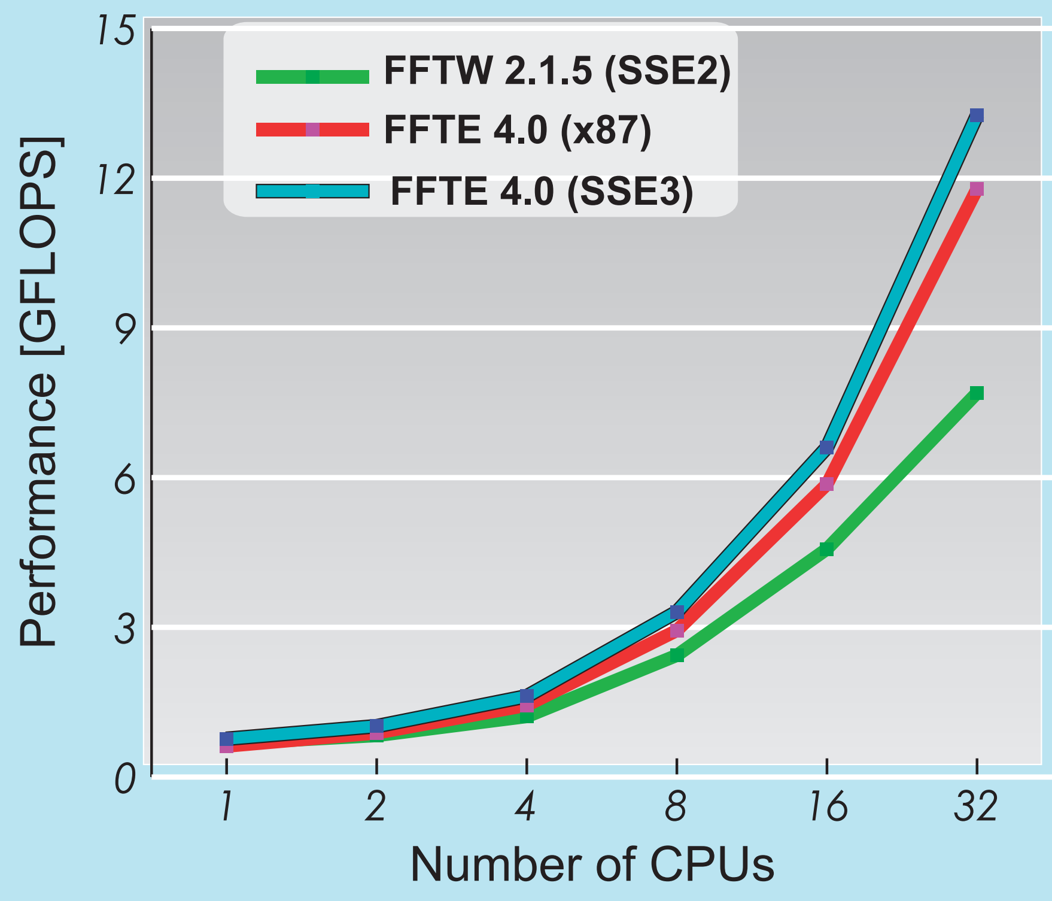
Approach

- Many FFT routines work well when data sets fit into a cache.
- When a problem size exceeds the cache size, however, the performance of these FFT routines decreases dramatically.
- Some previously presented six-step FFT algorithms require
  - Two multicolumn FFTs.
  - Three data transpositions.

  The chief bottlenecks in cache-based processors.
- We combine the multicolumn FFTs and transpositions to reduce the number of cache misses.

Performance of FFTE 4.0

Data:
N1 x N2 x N3 = 2^24 x P
Machines:
Xeon EM64T 3.0GHz
Gigabit Ethernet
1024 MB DDR2/400


Performance [GFLOPS]
Number of CPUs
FFTW 2.1.5 (SSE2)
FFTE 4.0 (x87)
FFTE 4.0 (SSE3)