Hierarchical Distributed Loop Self-Scheduling Schemes on Cluster and Cloud Systems

Dr. Anthony T. Chronopoulos,
Yiming Han

1 Introduction

2 Simple Loop Self-Scheduling Schemes

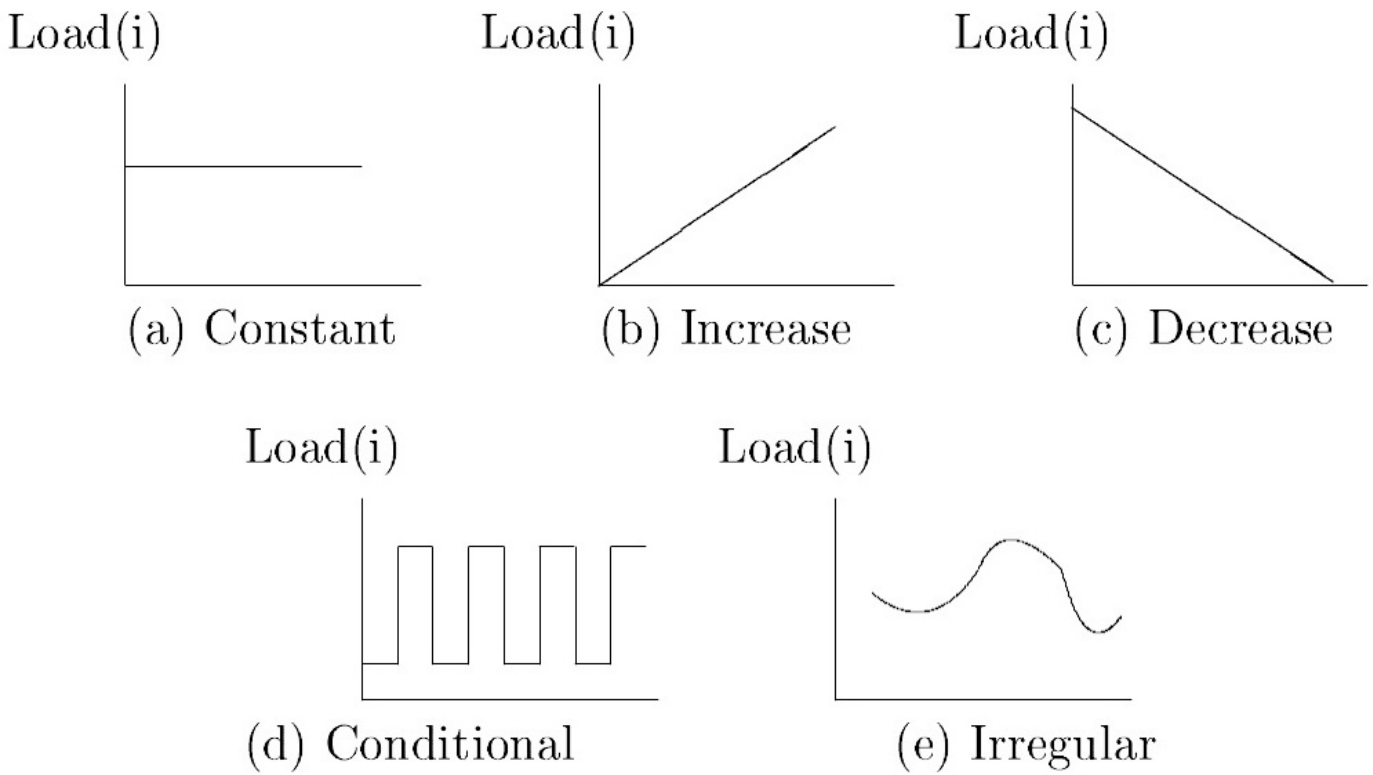
3 Distributed Loop Self-Scheduling Schemes

4 Hierarchical Distributed Loop Self-Scheduling Schemes

5 Conclusion and Future Work

Loops

1. Loop Styles: Constant, Increase, Decrease, Conditional, Irregular
2. Loop Scheduling: Static, Dynamic
3. Loop Carried Dependence: DOALL loops, DOACROSS loops



Examples of Parallel Loops

1. Mandelbrot Set

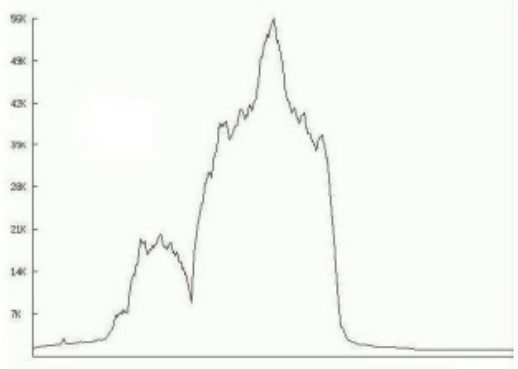
MSetLSM(MSet,nx,ny,xmin,xmax,ymin,ymax,maxiter)

```
BEGIN
    FOR iy = 0 TO ny - 1 DO
        cy = ymin + iy * (ymax - ymin)/(ny - 1)
        FOR ix = 0 TO nx - 1 DO
            cx = xmin + ix * (xmax - xmin)/(nx - 1)
            MSet[ix][iy] = MSetLevel(cx,cy,maxiter)
        END FOR
    END FOR
END
```

Examples of Parallel Loops

MSetLevel(cx,cy,maxiter)

```
BEGIN
    x = y = x2 = y2 = 0.0, iter = 0
    WHILE(iter < maxiter) AND (x2 + y2 < 2.0)DO
        temp = x2 - y2 + cx; y = 2 * x * y + cy;
        x = temp; x2 = x * x; y2 = y * y; iter++;
    END WHILE
    RETURN (iter)
END
```



Examples of Parallel Loops

1. Adjoint Convolution

```
BEGIN
    FOR I = 1 TO N  DO
        FOR J = I TO N  DO
            A(I) = A(I) + X * B(J) * C(J - I)
        END FOR
    END FOR
END
```

Goal: Minimization of Execution time

1. Assigning tasks to processors in order to make the processor loads well balanced
2. Avoiding assigning small number of iterations
3. Reducing communication overhead

Self-Scheduling Scheme

Common Solution:

1. Partition problem into several independent tasks (if possible)
2. Distribute workload on multiple machines
3. Collect results

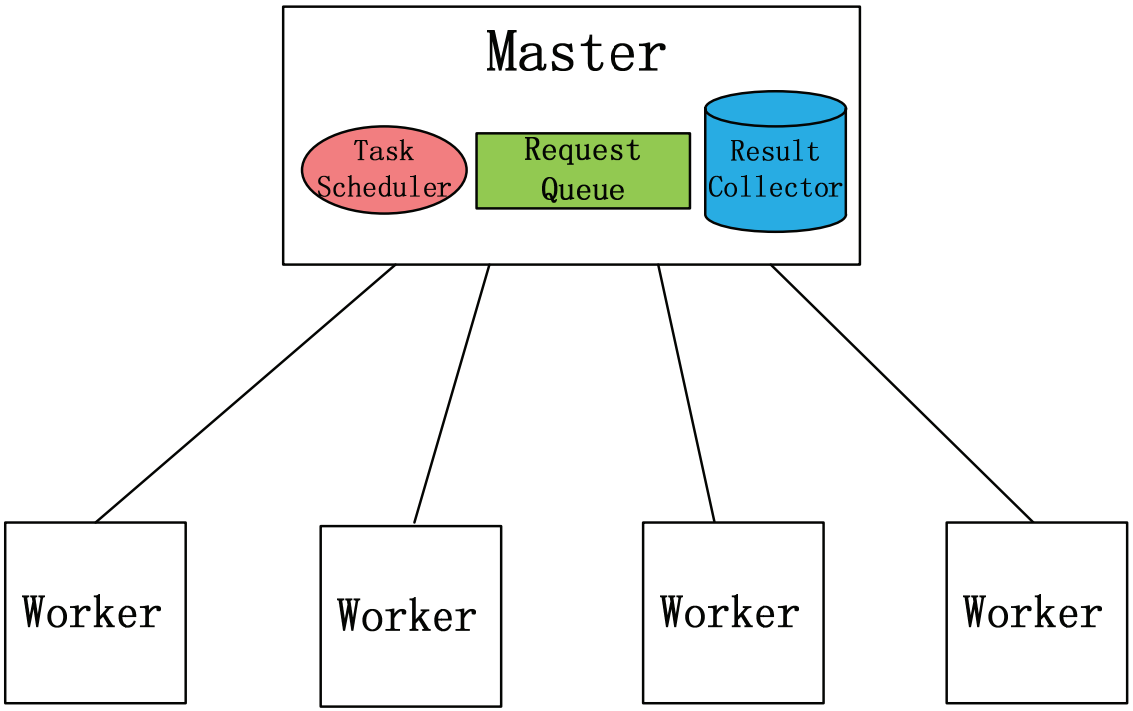


Figure : Self-Scheduling Schemes: the Master-Worker model

Simple Loop Scheduling Schemes

In a generic self-scheduling scheme, at the i_{th} scheduling step, the master computes the chunk-size C_i, a starting index i_{start}, and the remaining number of tasks R_i.

$$R_0 = I,\ C_i = f(R_{i-1}, p),\ i_{start} = J_0 \quad (1)$$

where $f(,)$ is a function possibly of more inputs than just R_{i-1} and p.

1. The master
   - Assigns to a worker C_i iterations and a starting index i_{start}
   - Then updates the i_{start} and R_i:

     $$i_{start} = i_{start} + C_i,\ R_i = R_{i-1} - C_i \quad (2)$$

     If the C_i is below a threshold, then computation of C_i must be modified by a *Threshold Condition*.

Simple Loop Scheduling Schemes

Algorithm of Simple Loop Scheduling Schemes:

1. Master:
   - Receive a new request from a worker for tasks
   - If($R_i > 0$) then
     - Compute C_i, i_{start} and R_i from (1)(2) above
     - Send a new task (starting index i_{start} and size C_i) to the worker
   - Else
     - Send 'terminate' signal to requesting workers
2. Worker:
   - Send a request to Master
   - Receive new tasks or 'terminate' signal
   - Perform tasks or terminate

Chunk Self-Scheduling (CSS)

1. $C_i = k$, where $k \geq 1$ and is chosen by the user.
   - $k = 1$: pure Self-Scheduling scheme (SS)
   - $k = I/p$: Fixed-size scheme (FS)
2. Strength: low scheduling overheads
3. Weakness: load imbalance, non-adaptive

Guided Self-Scheduling(GSS)

1. $C_i = \lceil R_{i-1}/p \rceil$
   - Dynamic
   - Large chunks at the beginning
   - Small chunks at the end
2. GSS(k)
3. Example:
   - $I = 1000$ and $p = 4$

Scheme	Chunk size
GSS	250 188 141 106 79 59 45 33 25 19 14 11 8 6 4 3 3 2 1 1 1 1

Factoring Self-Scheduling(FSS)

1. $C_i = \lceil R_{i-1}/(\alpha p) \rceil$, $R_i = R_{i-1} - pC_i$ (where $R_0 = I$)
   - Improve GSS
   - Better load balance
2. Example:
   - $I = 1000$ and $p = 4$

Scheme	Chunk size
FSS	125 125 125 125 63 63 63 63 31 31 31 31 16 16 16 16 8 8 8 8 4 4 4 4 2 2 2 2 1 1 1 1

Trapezoid Self-Scheduling (TSS)

1. $F = \left\lfloor \frac{I}{2p} \right\rfloor = 125,\ L = 1,\ N = \left\lceil \frac{2*I}{(F+L)} \right\rceil = 16$
2. $D = \left\lfloor \frac{(F-L)}{(N-1)} \right\rfloor = 8$
3. $C_i = C_{i-1} - D$,
   - reasonable load balance
   - Less synchronization
4. Example:
   - $I = 1000$ and $p = 4$

Scheme	Chunk size
TSS	125 117 109 101 93 85 77 69 61 53 45 37 28

Heterogeneity

1. heterogeneity
   - heterogeneous program has parallel loops with different amount of work in each iteration;
     - Load Balance
   - heterogeneous processors have different speeds;
     - Load Balance
   - a heterogeneous network has different cost of communication between processors.
     - Synchronization
     - Communication

Cloud Computing

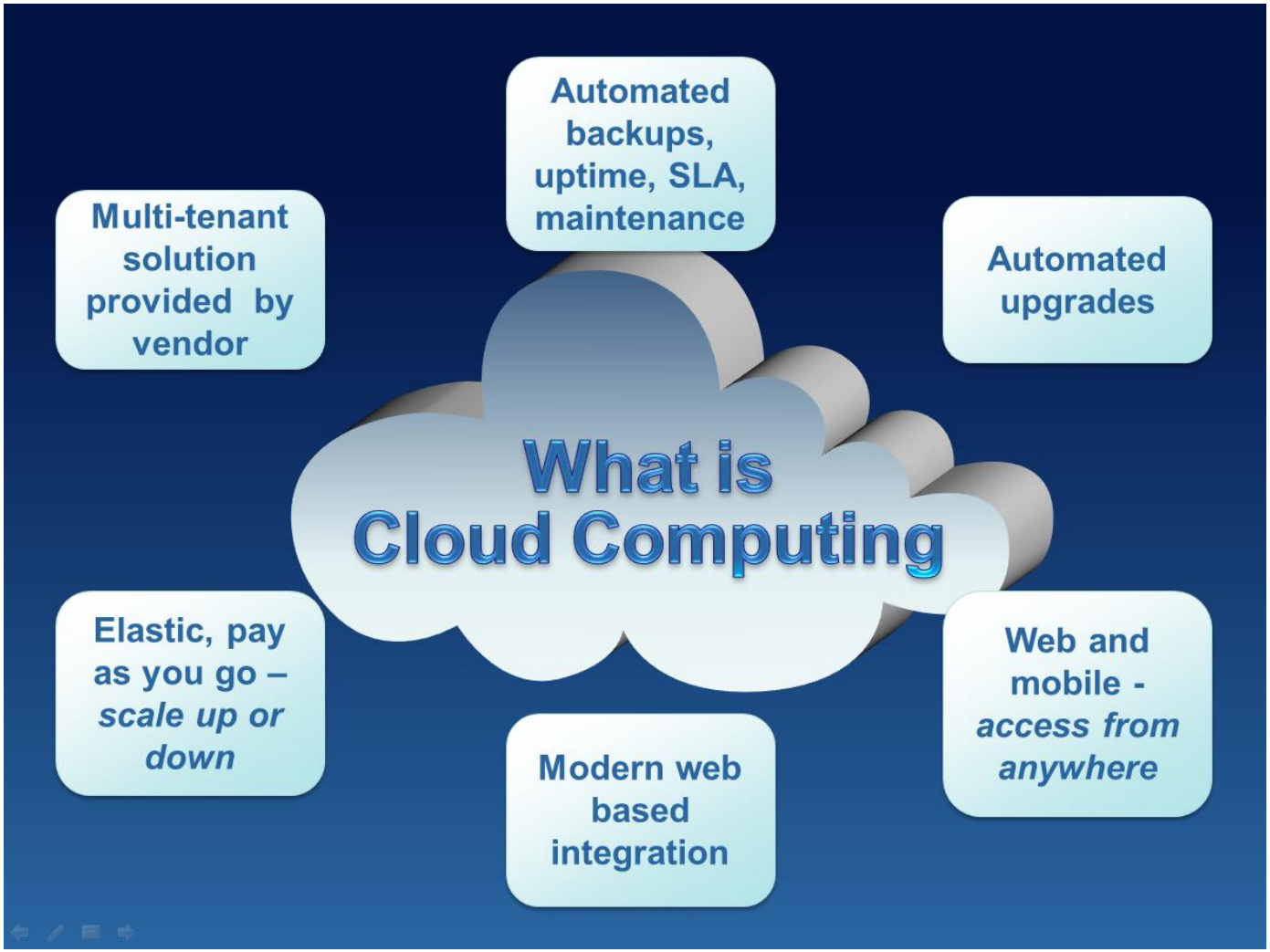
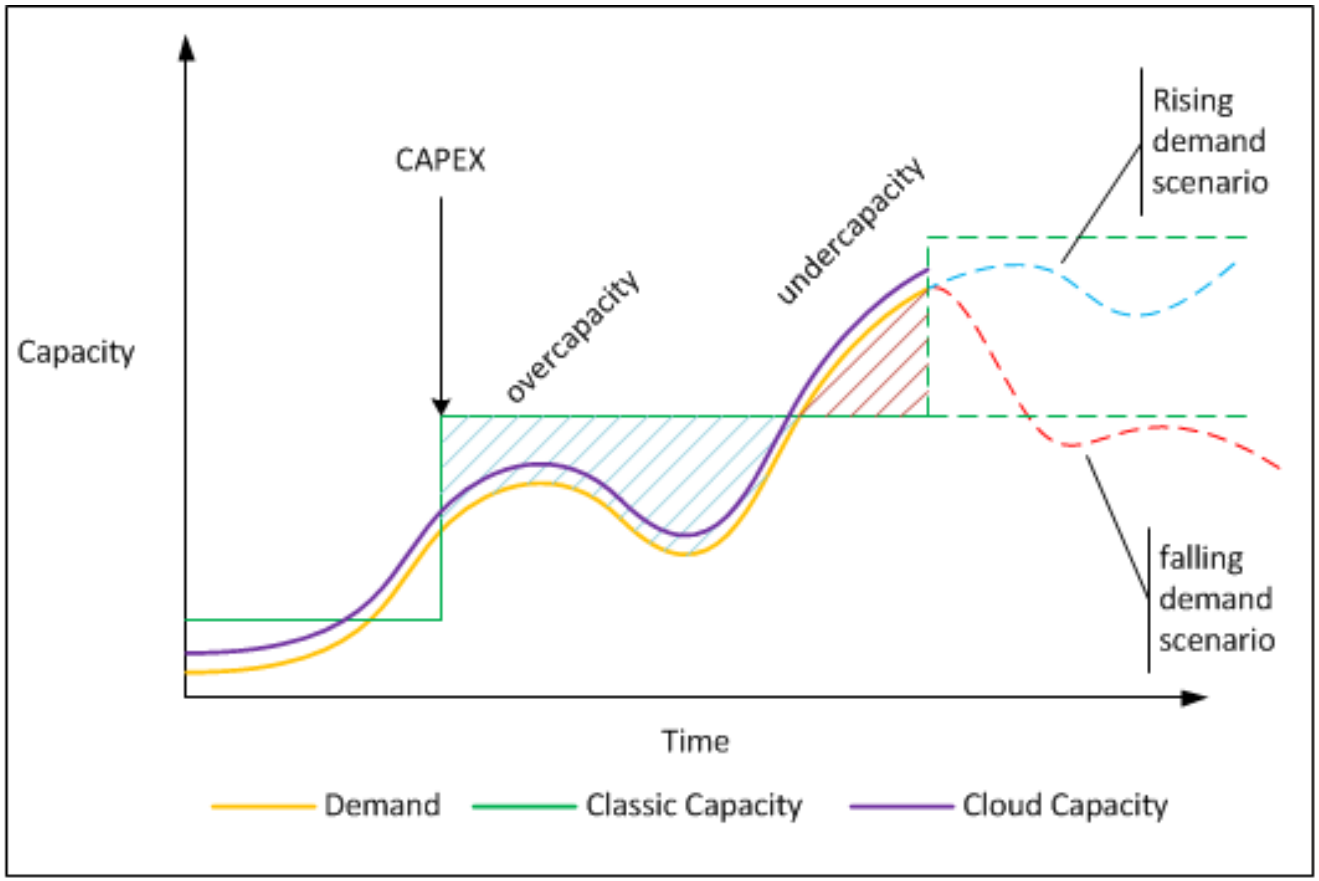


Figure : What is Cloud Computing (from http://abouttmc.com/)

Cloud Computing Characteristics

1. On-demand self-service



Cloud Computing Characteristics

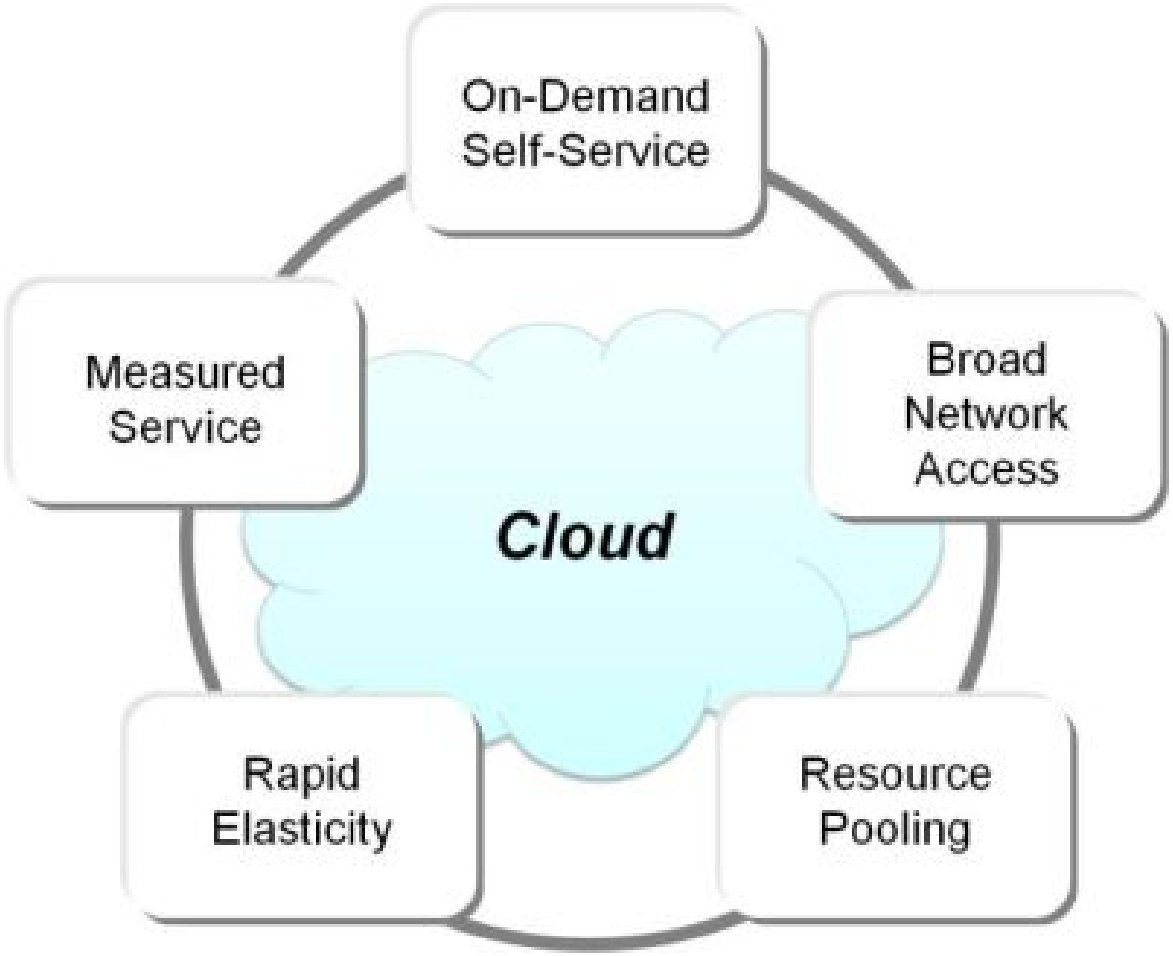


Figure : Cloud Computing Characteristics (from http://www.opengroup.org/)

Cloud Service Models

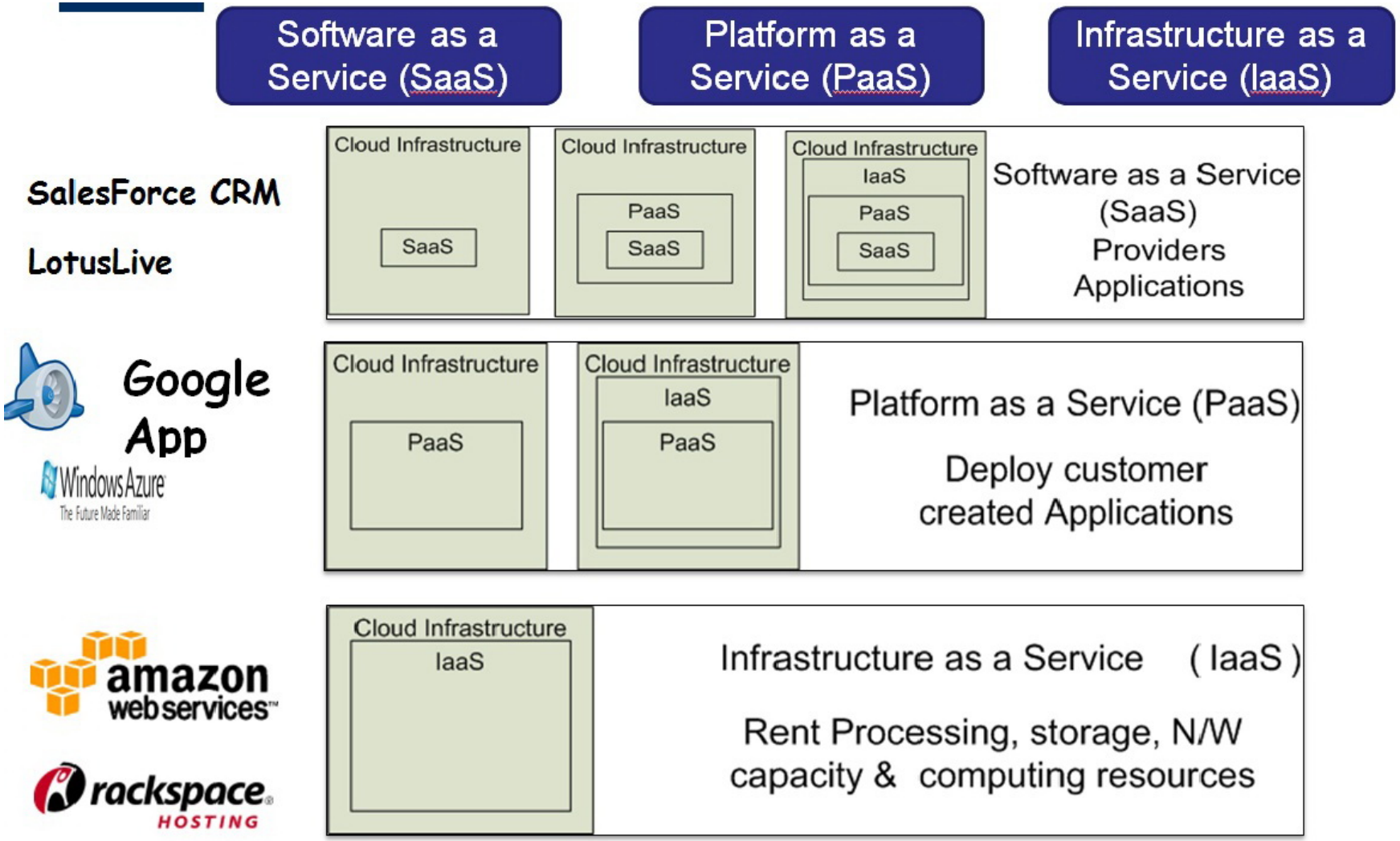


Figure : Cloud Service Models (from Mark Baker's lecture)

Heterogeneous

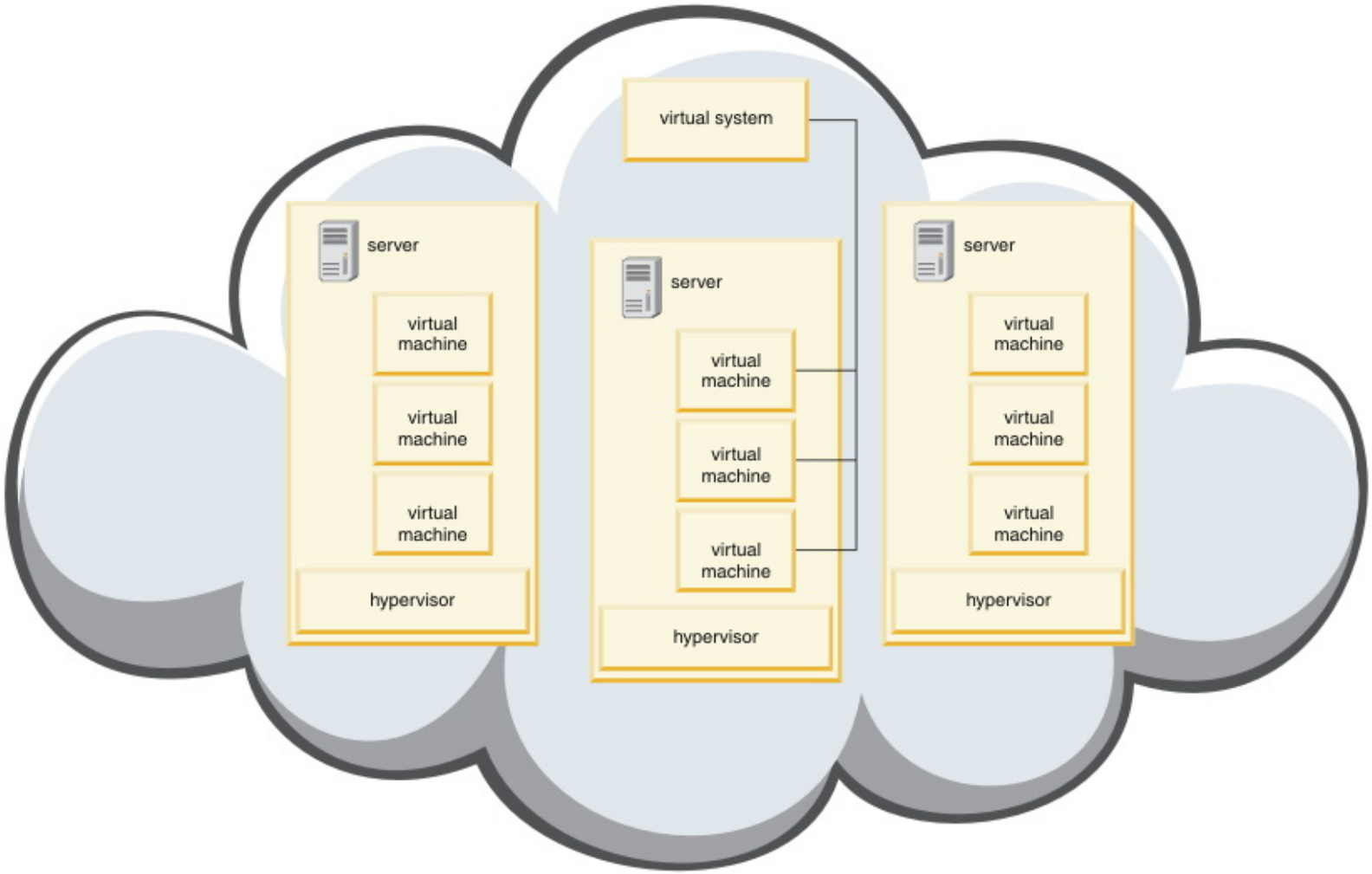


Figure : virtual system (from http://pic.dhe.ibm.com/)

Algorithm

1. $V_j = Speed(P_j)/min_{1 \le i \le p}\{Speed(P_i)\}$, $j = 1, ..., p$, is the virtual power of P_j (computed by the master), where Speed(P_j) is the processing speed of P_j. That is a standardized computing power.
2. $V = \sum_{j=1}^{p} V_j$ is the total virtual computing power.
3. DC is the distributed chunk size for one worker request.
4. Example, DGSS, Initialization: $R = I$:

Algorithm 1 Calculate DC, DGSS

for $i = 1$ **to** V_j **do**
  $DC = DC + \lceil R/V \rceil$;
  $R = R - \lceil R/V \rceil$;
end for
return DC;

Experiment

1. Applications
   - Mandelbrot Set
     - Size: 10K * 10K, 20K * 20K
   - Adjoint Convolution
     - Size: 9K * 9K, 16K * 16K
2. Platform
   - FlexCloud of Institute for Cyber Security(ICS)
     - 5 Racks of Dell R410, R610, R710, and R910s consisting of 748 processing cores, 3.44TB of RAM, and 144TB of total storage.
     - Redundant 10GB network connectivity provides high performance access between all nodes.
     - Joynet

Experiment

1. Experimental Setup
   - 17 VMs, single core, 1GB memory, 10GB storage
   - Ubuntu Linux 10.04 image.
   - GCC/G++ and OpenMPI are installed
   - Stress
     - Configure CPU, memory, I/O, and disk stress
     - apt-get install stress
     - 8, 5, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.

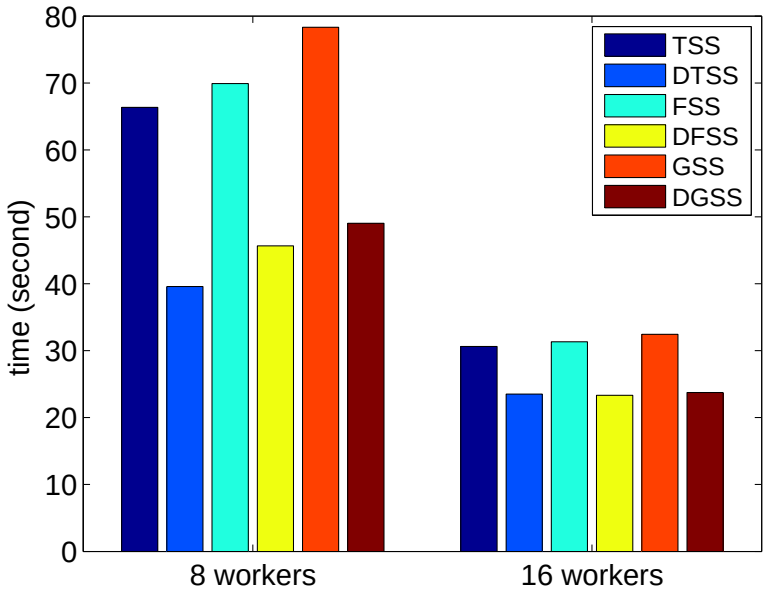
Experiment

Stress

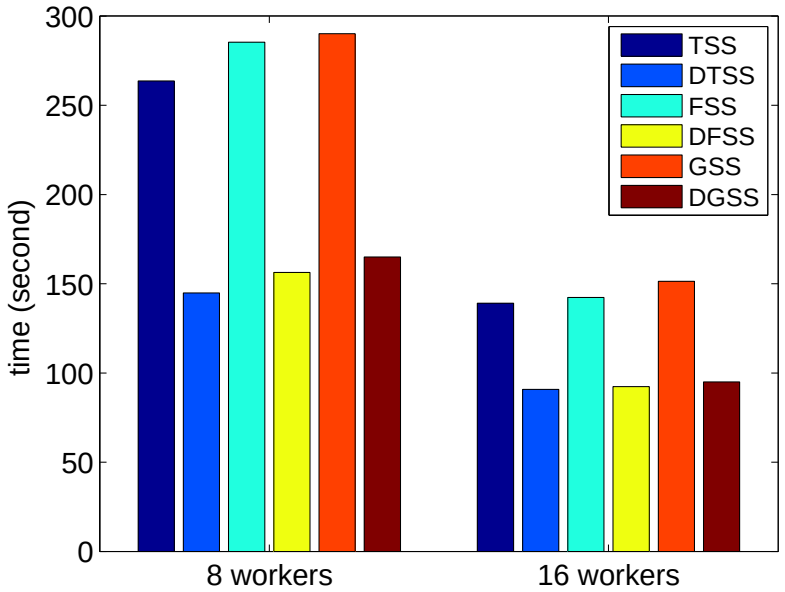
```
yhan@ubuntu: ~
-c: command not found
yhan@ubuntu:~$ stress
`stress' imposes certain types of compute stress on your system

Usage: stress [OPTION [ARG]] ...
 -?, --help         show this help statement
     --version      show version statement
 -v, --verbose      be verbose
 -q, --quiet        be quiet
 -n, --dry-run      show what would have been done
 -t, --timeout N    timeout after N seconds
     --backoff N    wait factor of N microseconds before work starts
 -c, --cpu N        spawn N workers spinning on sqrt()
 -i, --io N         spawn N workers spinning on sync()
 -m, --vm N         spawn N workers spinning on malloc()/free()
     --vm-bytes B   malloc B bytes per vm worker (default is 256MB)
     --vm-stride B  touch a byte every B bytes (default is 4096)
     --vm-hang N    sleep N secs before free (default is none, 0 is inf)
     --vm-keep      redirty memory instead of freeing and reallocating
 -d, --hdd N        spawn N workers spinning on write()/unlink()
     --hdd-bytes B  write B bytes per hdd worker (default is 1GB)
     --hdd-noclean  do not unlink files created by hdd workers

Example: stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 10s
```

Results (execution time)

1. Mandelbrot Set execution time, seconds



a. 10K * 10K

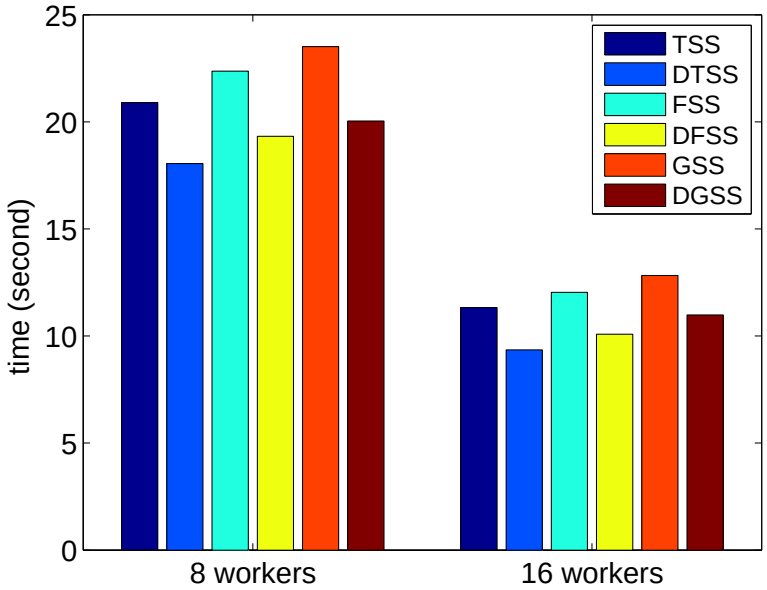


b. 20K * 20K

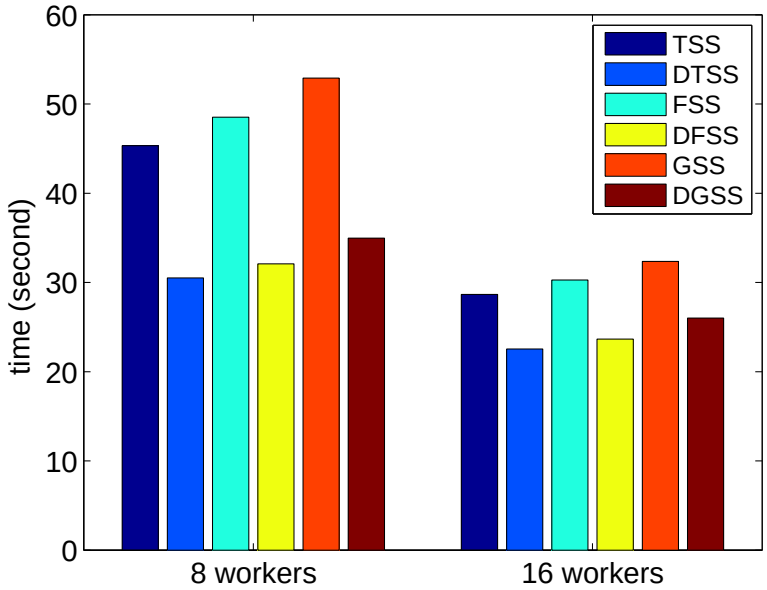
Figure : The performance comparison of Mandelbrot Set using distributed schemes

Results (execution time)

1. Adjoint Convolution execution time, seconds



a. 9K * 9K

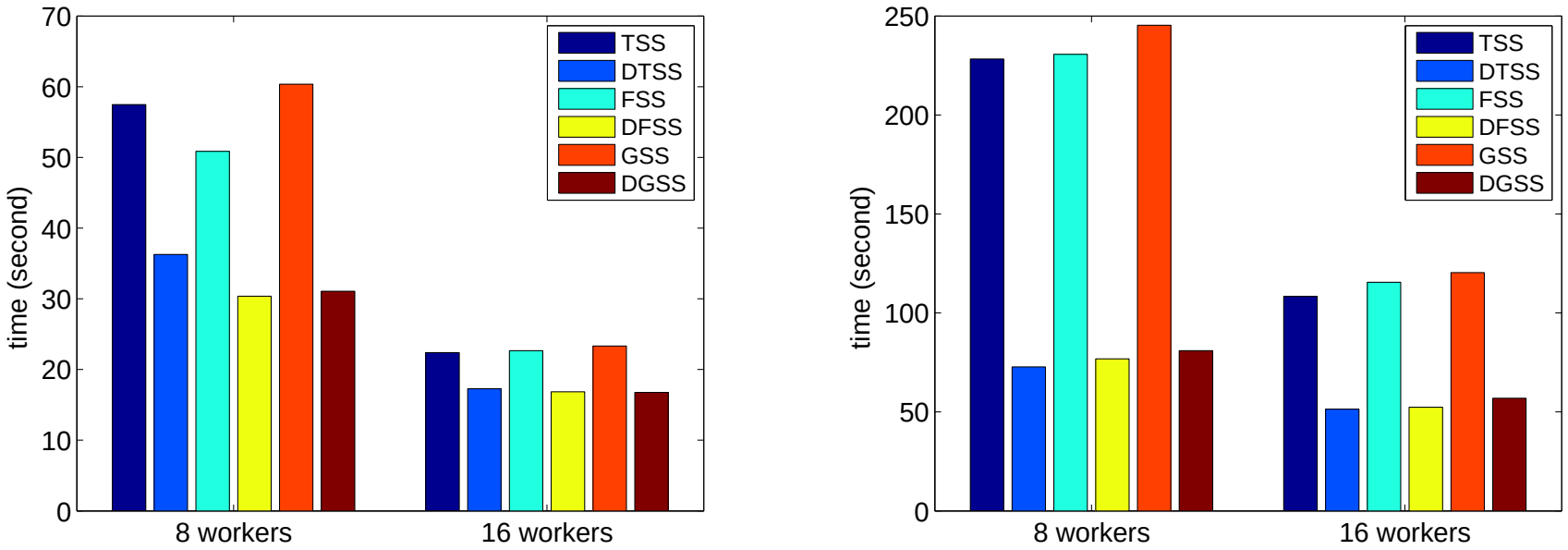


b. 16K * 16K

Figure : The performance comparison of Adjoint Convolution using distributed schemes

Results (*maximum difference*)

1. *maximum difference* = $max\{T_{comp_1}, T_{comp_2}, \ldots, T_{comp_p}\} - min\{T_{comp_1}, T_{comp_2}, \ldots, T_{comp_p}\}$
2. Mandelbrot Set *maximum difference*, seconds



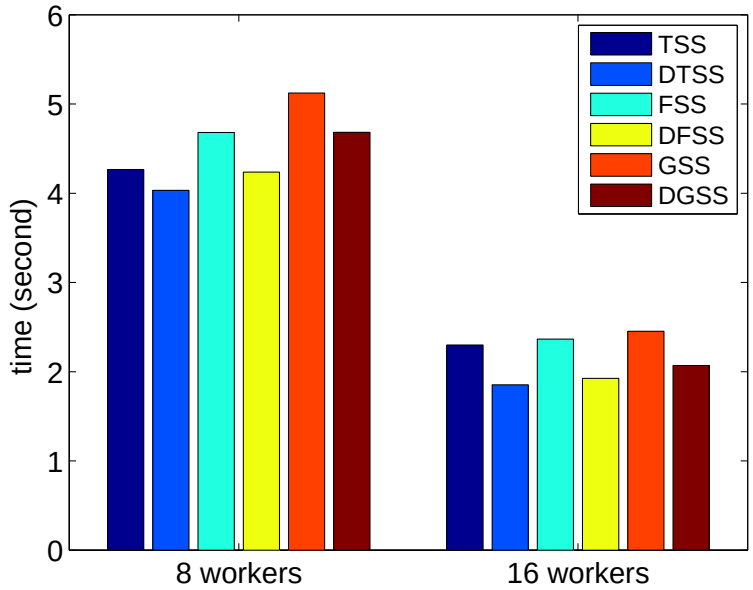
a. 10K * 10K

b. 20K * 20K

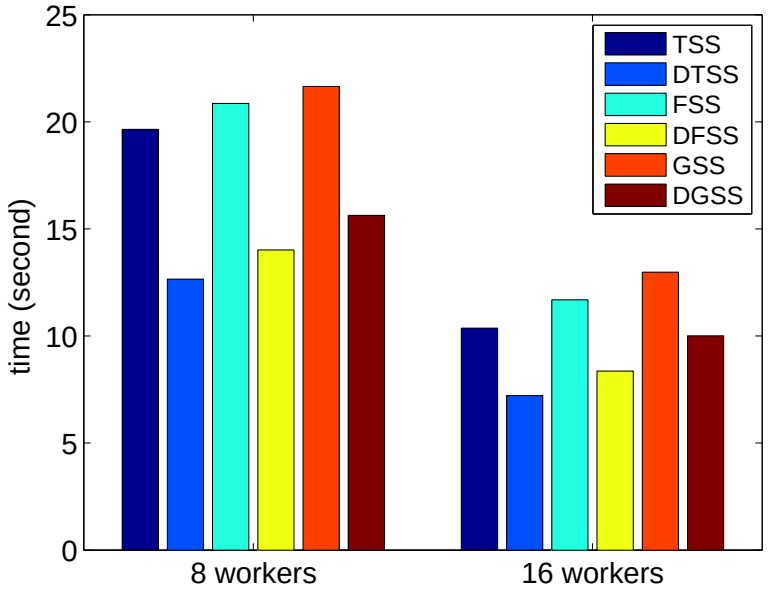
Figure : The *maximum difference* comparison of Mandelbrot Set using distributed schemes

Results (*maximum difference*)

1. Adjoint Convolution *maximum difference*, seconds



a. 9K * 9K

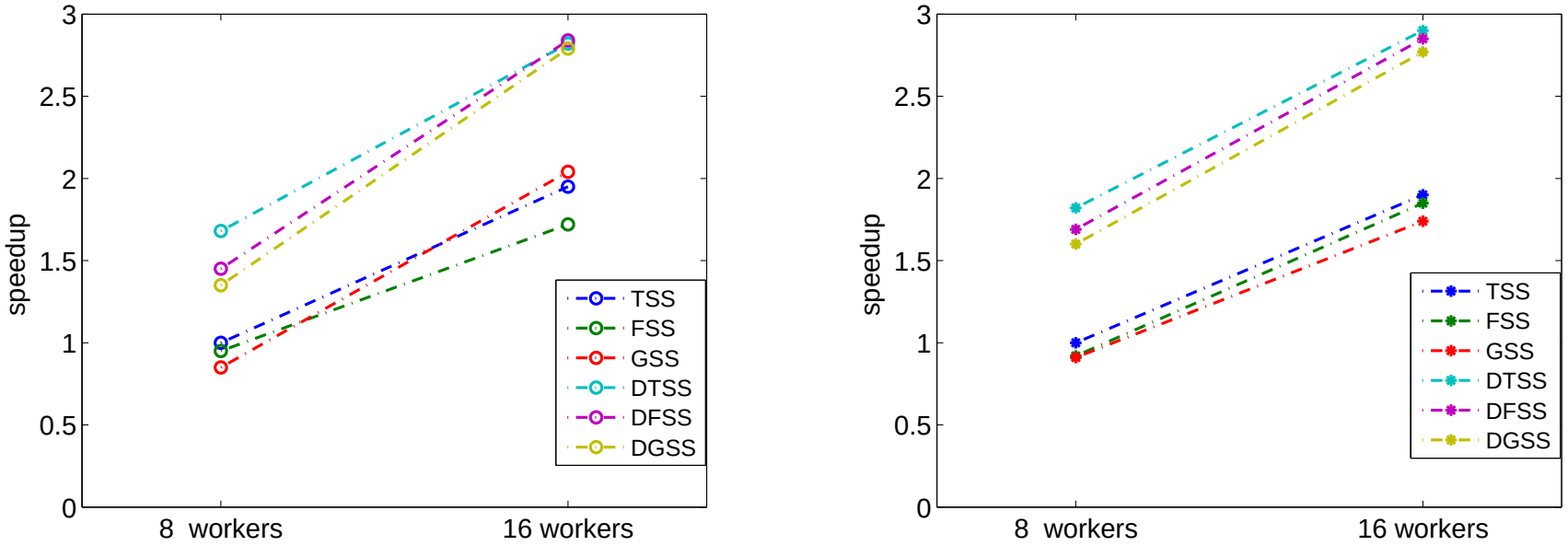


b. 16K * 16K

Figure : The *maximum difference* comparison of Adjoint Convolution using distributed schemes

Results (speedup)

1. Baseline: $\hat{T}_1$ is the execution time for the TSS with 8 workers
2. Speedup: $S_p = \frac{\hat{T}_1}{T_p}$
3. Mandelbrot Set speedup



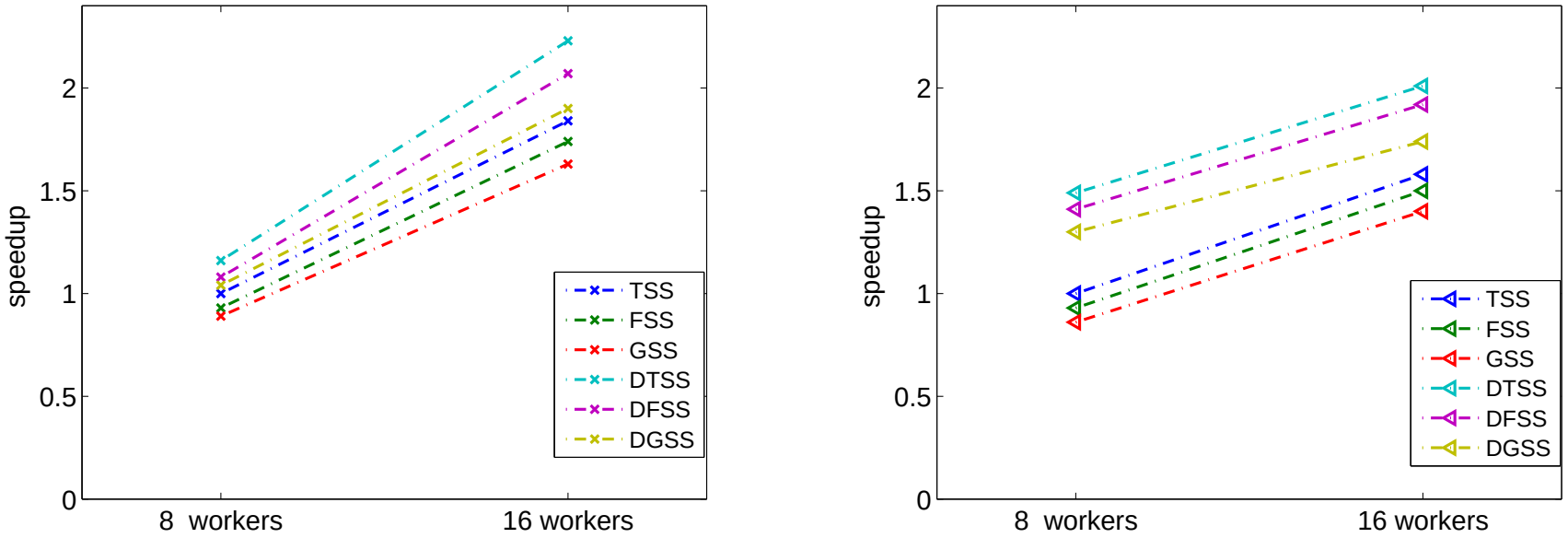
a. 10K * 10K

b. 20K * 20K

Figure : The speedup comparison of Mandelbrot Set using distributed schemes

Results (speedup)

1. Baseline: $\hat{T}_1$ is the execution time for the TSS with 8 workers
2. Speedup: $S_p = \frac{\hat{T}_1}{T_p}$
3. Adjoint Convolution speedup



a. 9K * 9K

b. 16K * 16K

Figure : The speedup comparison of Adjoint Convolution using distributed schemes

Hierarchical Schemes on Large-Scale Cluster

1. Master-Worker Model
   - Centralized: one master, multiple workers
   - Easy to implement and maintain
   - good performance
   - Not scalable

Hierarchical Schemes on Large-Scale Cluster

1. Scalability

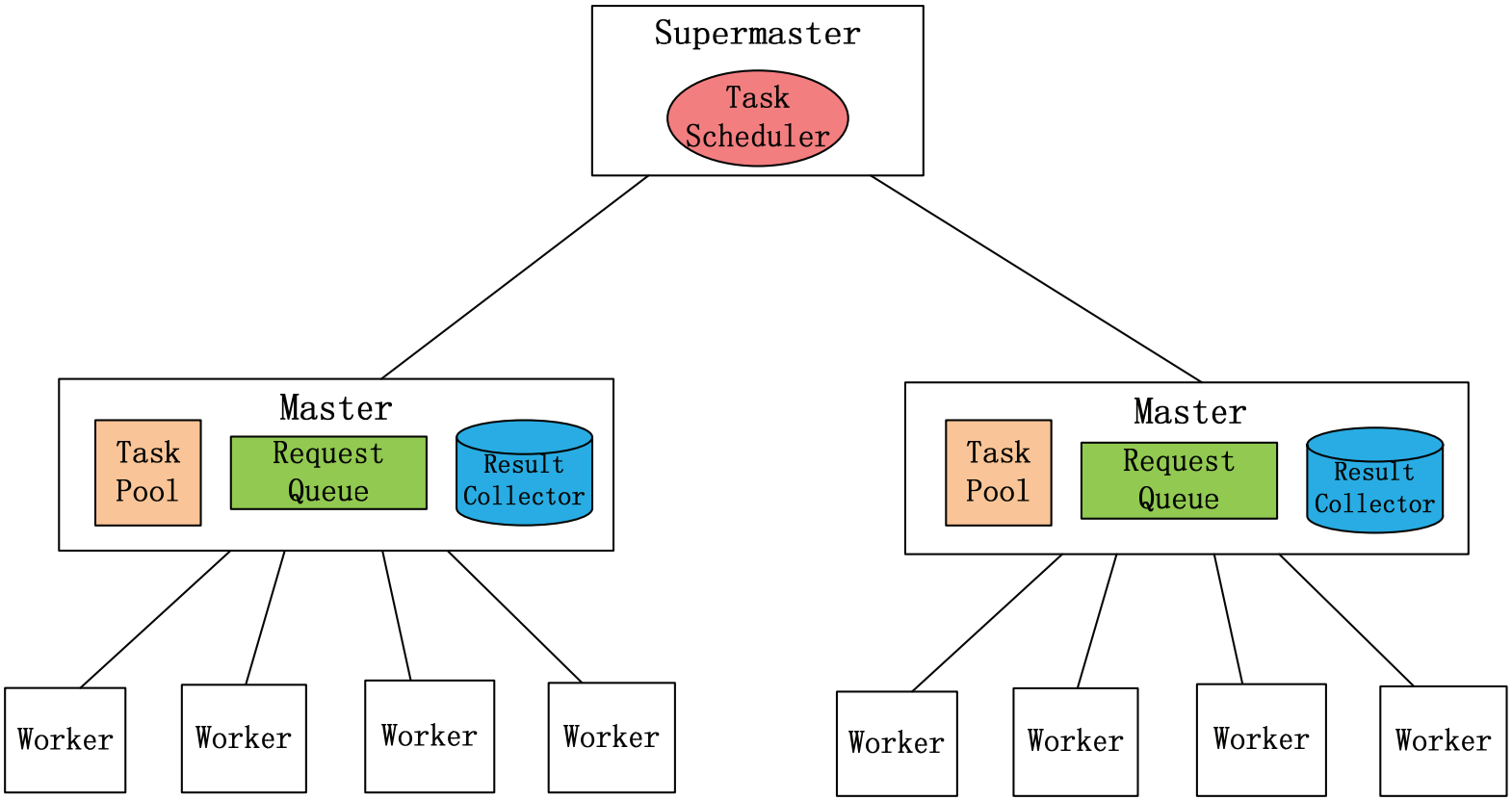


Figure : Hierarchical Model

Experiment

1. Applications
   - Mandelbrot Set
     - Size: 200K * 200K
   - Adjoint Convolution
     - Size: 640K * 640K
2. Platform
   - Ranger, Texas Advanced Computing Center(TACC), UTexas at Austin
     - SunBlade x6420 blade, 4 AMD Opteron Quad-Core 64-bit processors, 16 cores totally.
     - total of 62,976 compute cores, 123 TB of total memory and 1.7 PB of raw global disk space.
     - InfiniBand Constellation Core Switch.
     - Large option.

Ranger

Figure : Ranger (from http://www.tacc.utexas.edu/)

Results (execution time)

1. Mandelbrot Set execution time, seconds

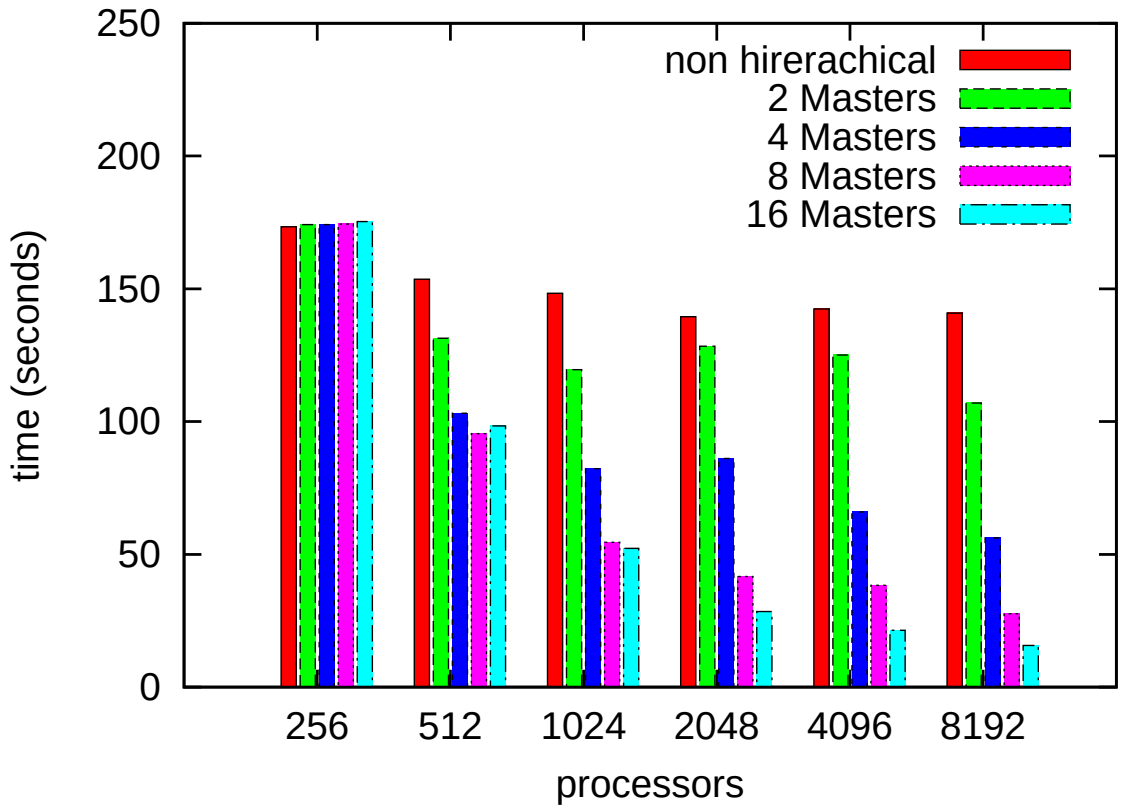


Figure : The performance of Mandelbrot Set using hierarchical distributed schemes

Results (execution time)

1. Adjoint Convolution execution time, seconds

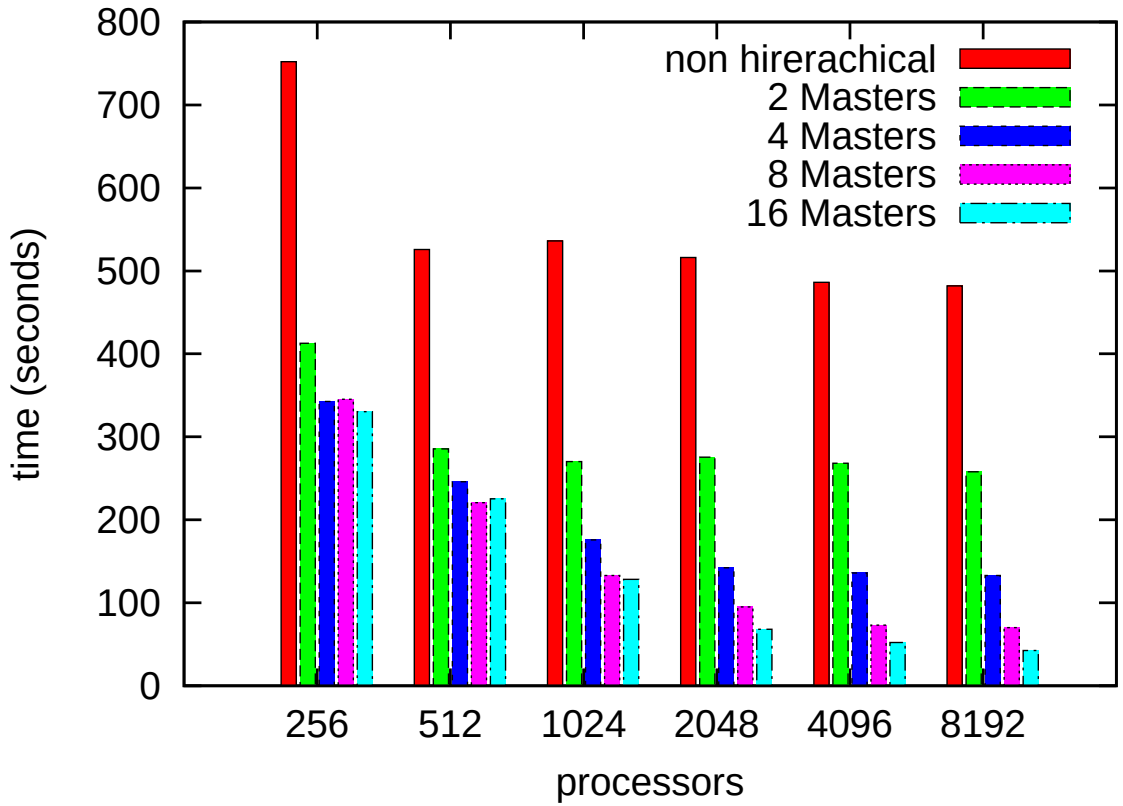
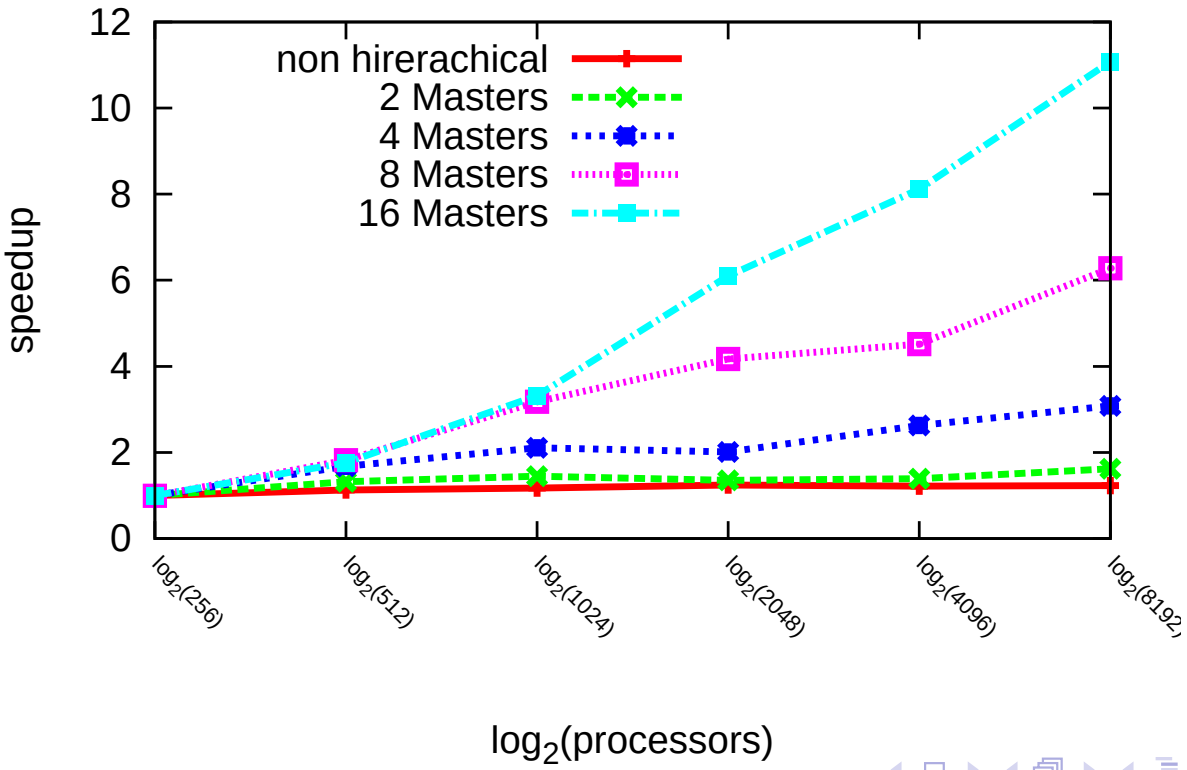


Figure : The performance of Adjoint Convolution using hierarchical distributed schemes

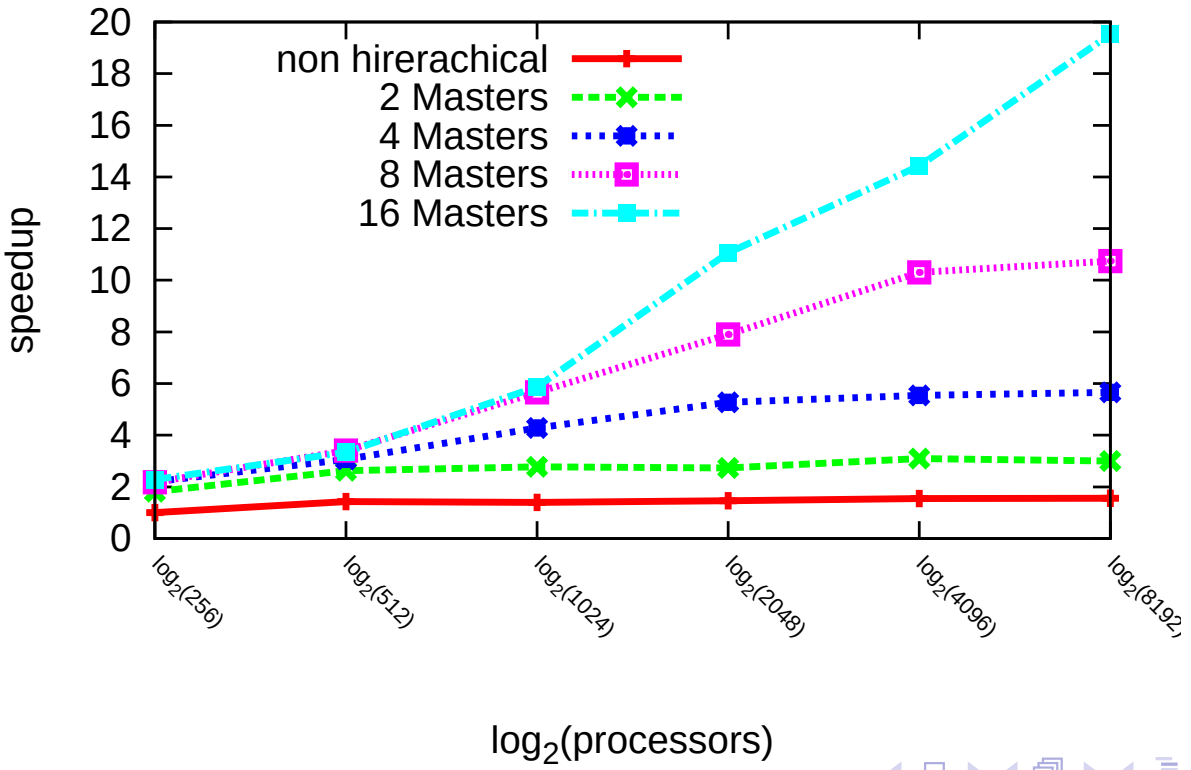
Results (speedup)

1. Baseline: $\hat{T}_1$ is the execution time for non hierarchical distributed scheme with 256 workers
2. Speedup: $S_p = \frac{\hat{T}_1}{T_p}$
3. Mandelbrot Set speedup



Results (speedup)

1. Baseline: $\hat{T}_1$ is the execution time for non hierarchical distributed scheme with 256 workers
2. Speedup: $S_p = \frac{\hat{T}_1}{T_p}$
3. Adjoint Convolution speedup



Conclusion

1. Simple Loop Self-Scheduling Schemes
   - Master-Worker Model
   - CSS
   - GSS
   - FSS
   - TSS
2. Distributed Loop Self-Scheduling Schemes
   - Load Balance
3. Hierarchical Distributed Loop Self-Scheduling Schemes
   - Scalability
     - Synchronization
     - Communication

Future Work

1. Discover schemes for DOACROSS loops
2. Apply Hierarchical Distributed Loop Self-Scheduling Schemes on cloud systems
   - Network heterogeneity
3. Hadoop MapReduce
   - Implement different schemes
   - Tune performance with different configurations

References

1. Y. Han, A. T. Chronopoulos, A Hierarchical Distributed Loop Self-Scheduling Scheme for Cloud Systems, Proceedings of IEEE NCA 2013, The 12th IEEE International Symposium on Network Computing and Applications, pp. 7-10, Boston, MA, USA, August 2013.
2. Y. Han, A. T. Chronopoulos, Distributed Loop Scheduling Schemes for Cloud Systems, Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium, High-Performance Grid and Cloud Computing Workshop, pp. 955-962, Boston, Massachusetts, USA, May 2013.