HPC Architecture

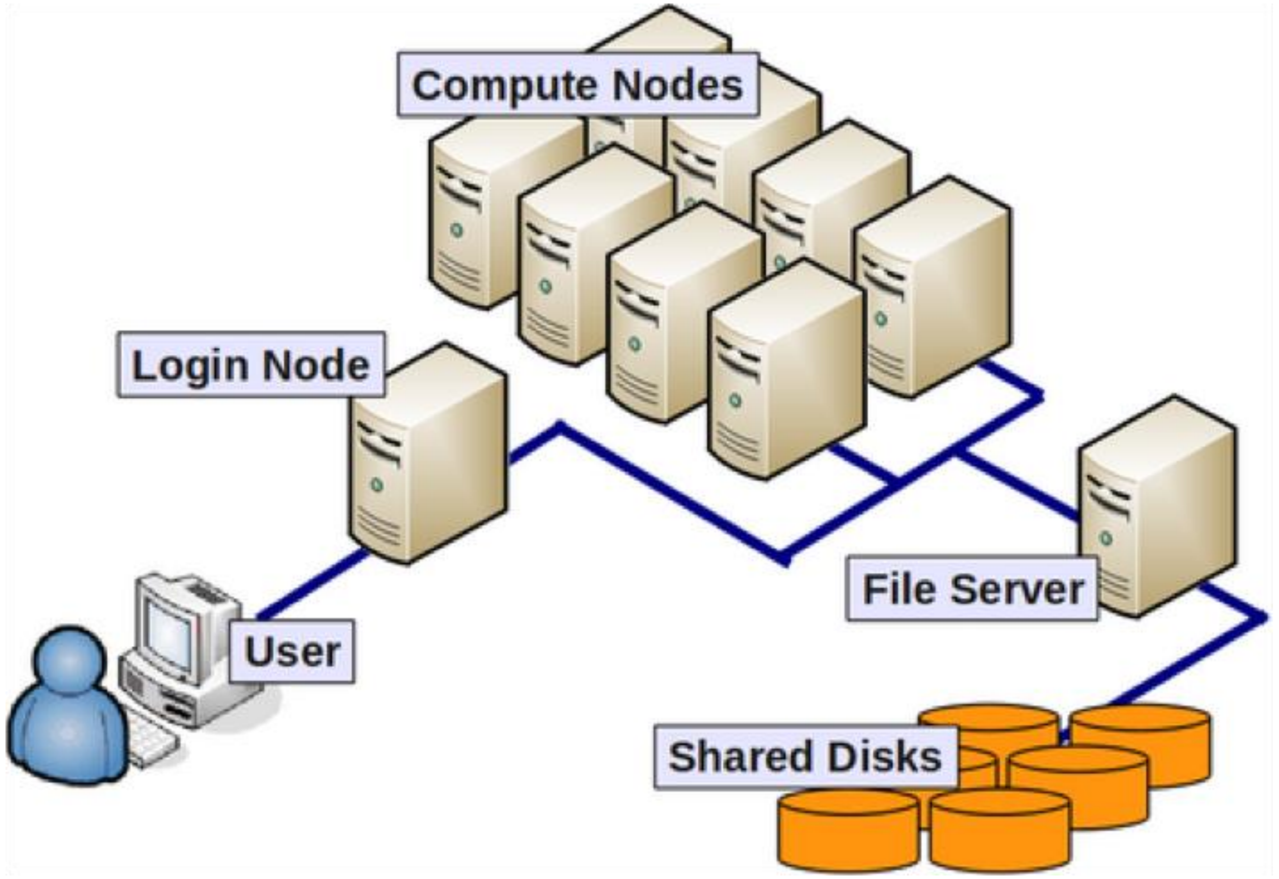
Shamjith K V, Lochan Kedkar, Abhishek Patil

Overview

- HPC Job Scheduling,
- Performance analysis
- Debugging
- Profiling,
- Petascale and Exascale Computing

HPC Job Scheduling

HPC Cluster



Cluster Terminology

- Supercomputer/High Performance Computing (HPC) cluster: A collection of similar computers connected by a high speed interconnect that can act in concert with each other.
- Server, Node, Blade, Box, Machine : An individual motherboard with CPU, memory, network and local hard drive.
- CPU (Socket): Central Processing Unit, a single silicon die that can contain multiple computational cores
- Core: Basic unit of compute that runs a single instruction of code
- GPGPU: General Purpose Graphics Processing Unit, a GPU designed for supercomputing.
- InfiniBand (IB): A near zero latency high bandwidth interconnect used in Supercomputing
- Serial: Doing tasks/instructions in sequence on a single core
- Parallel: Doing tasks/instructions on multiple cores simultaneously
- I/O: Input/Output, a general term for reading and writing files to/from storage whether local or remote.

What is a Job

- Job
  - user’s program/name of an executable.
  - input data and parameters
  - environment variables
  - required libraries
  - descriptions of computing resources required
- Job Script
  - Formal specification
  - identifies an application to run along with its input data and environment variables
  - requests computing resources

Local Resource Manager(LRM)/Batch System

- Job Scheduler or Workload Manager
  - Identifies jobs to run, selects the resources for the job, and decides when to run the job
- Resource Manager
  - identifies the compute resources and keeps track of their usage and feeds back this information to the workload manager
- Execution Manager
  - job initiation and start of execution is co-ordinated by the execution manager of the batch system.

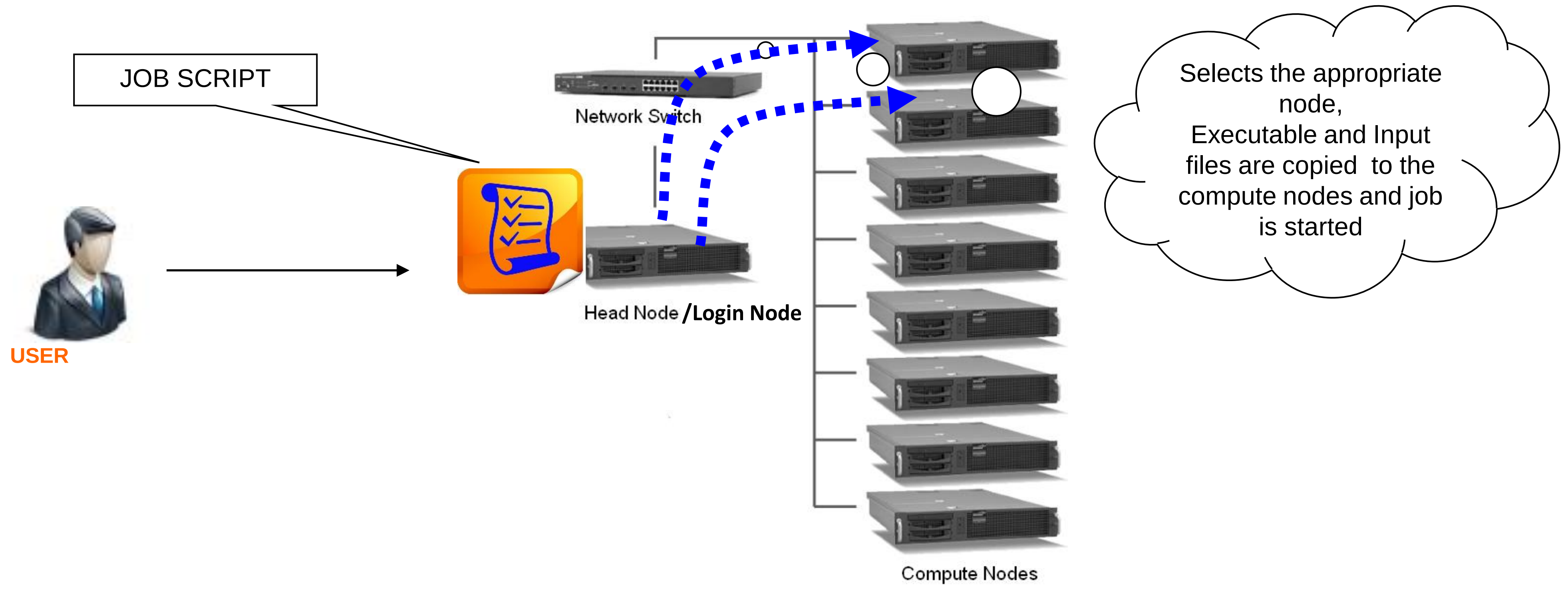
Job Scheduling

- The LRM is responsible for receiving and parsing the job script.
- If a job cannot be executed immediately, it is added to a queue.
- The job waits in the queue until the job's requested resources are available.
- The LRM then runs the job, when the requested resources are available

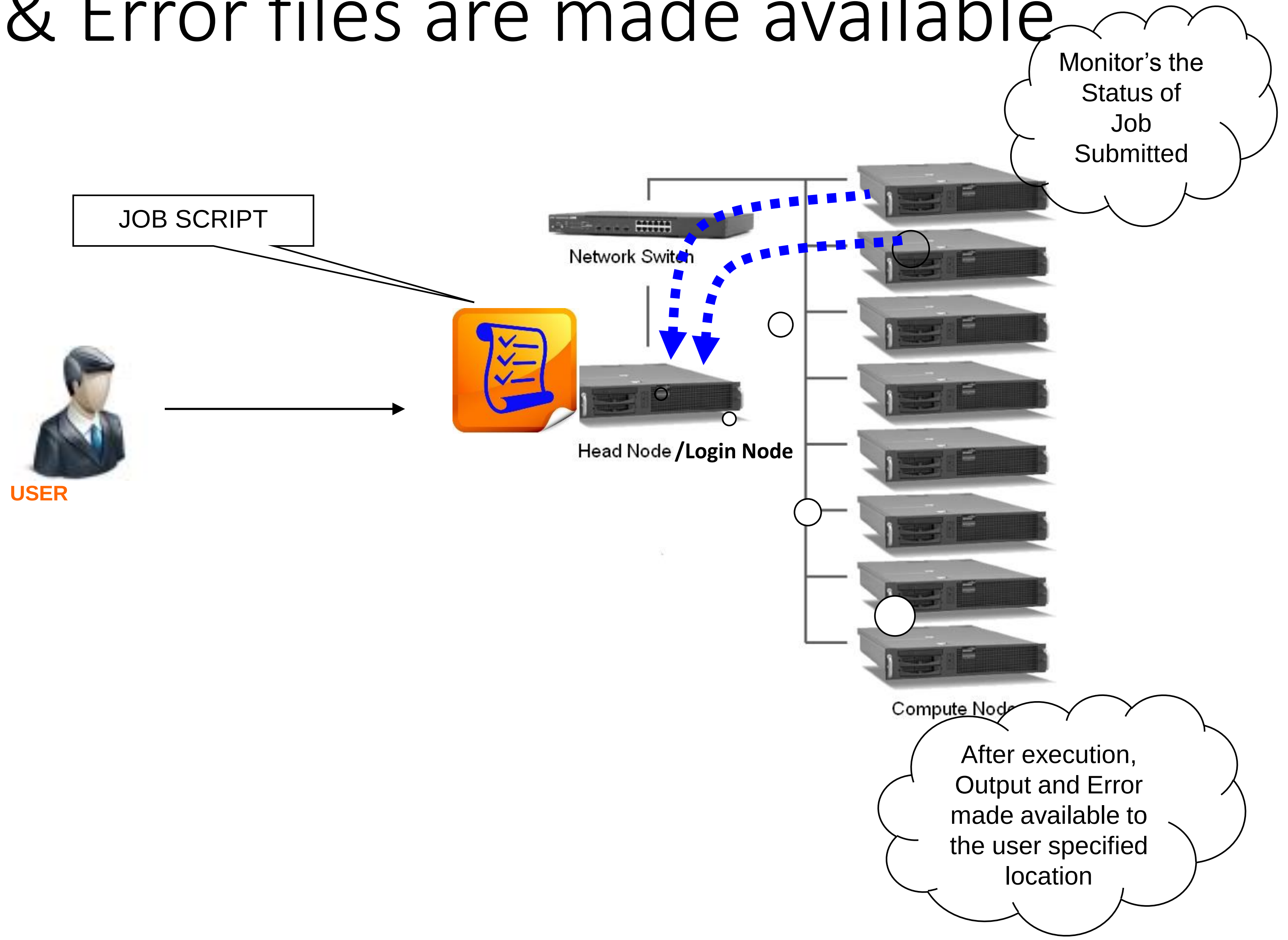
Job Scheduling policies

- FCFS
  - Latest job submitted is added to the bottom of the queue.
  - No other job in the queue will run before the job that is at the top of the queue.
  - Top job waits in the queue until enough jobs finish to free up the resources that it needs.
- Multi-priority queues
  - Clusters with heterogeneous resources or job mixes configure multiple job queues for separation
  - Queues can be defined to support different job sizes, schedule to specific cluster resources
  - Queues can have different priorities to allow different categories of jobs to be scheduled differently
- Back-filling:
  - If the scheduler has the intelligence to launch jobs lower in the queue, on resources that are currently idle, it is called a back-fill scheduler.
  - The back-fill scheduler follows a strict rule to only schedule lower priority jobs on idle resources if it will not delay the start of the top priority job.
- Fair-share:
  - A method to allocate resource shares to a user or groups of users or a project
  - A fair method for ordering jobs based on their usage history or criteria based on pricing
  - The job to be run next is selected from the set of jobs belonging to the most deserving entity
- Preemptive:
  - A job with higher priority can signal currently running job to stop and release resources to allow the high priority job to run.
  - Necessary requirement for pre-emption is support for job checkpoints and restart ability.

Job Execution in Compute Nodes


JOB SCRIPT
USER
Network Switch
Head Node /Login Node
Compute Nodes
Selects the appropriate node,
Executable and Input files are copied to the compute nodes and job is started

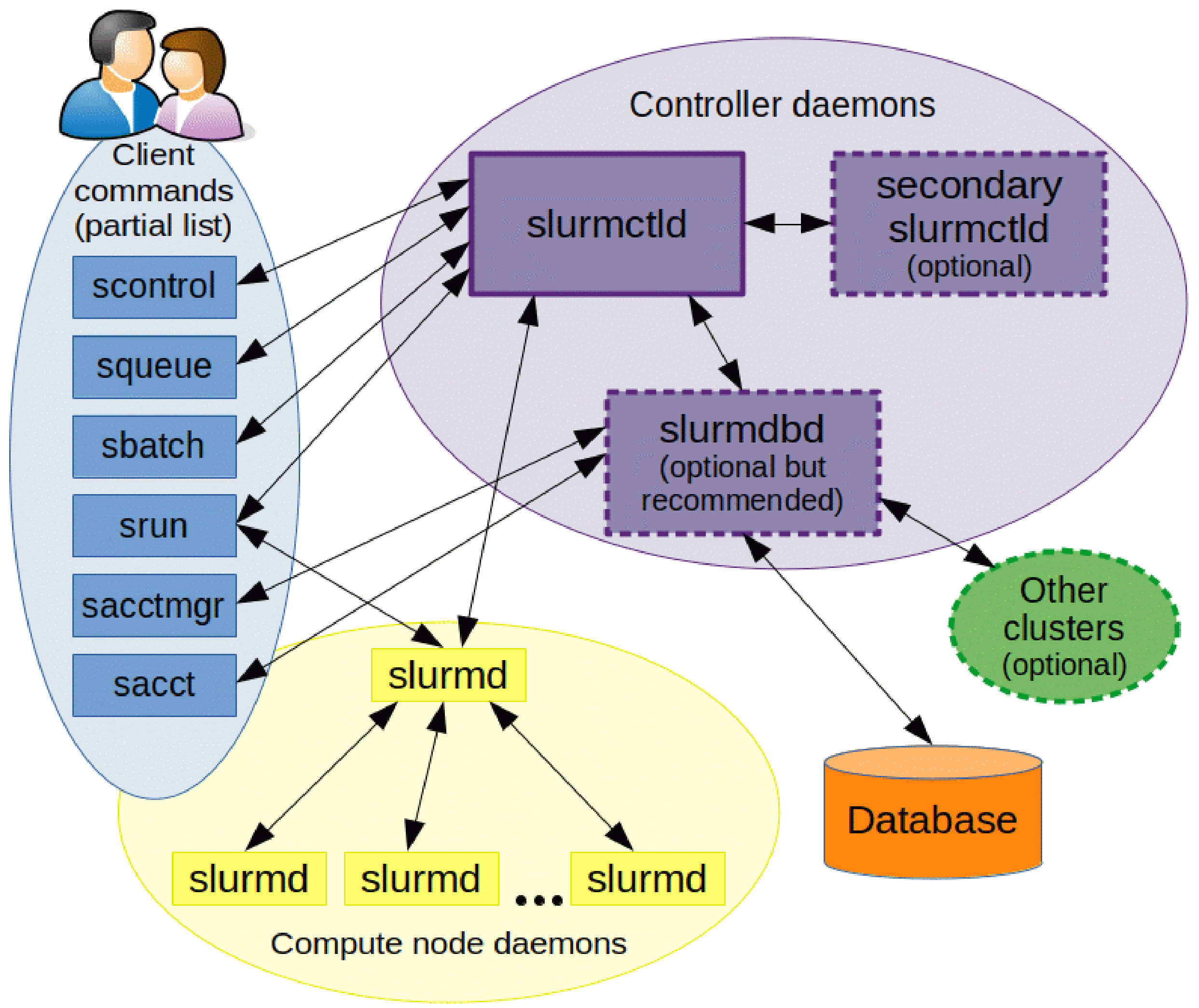
Output & Error files are made available


Monitor’s the Status of Job Submitted
JOB SCRIPT
Network Switch
Head Node /Login Node
USER
Compute Nodes
After execution, Output and Error made available to the user specified location

Local Resource Manager - SLURM

- ➢ When you login to HPC cluster, you land on Login Nodes
  - ○ Login nodes are not meant to run jobs
  - ○ These are used to submit jobs to Compute Nodes.
- ➢ To submit job on the cluster, you need to write a scheduler job script
- ➢ SLURM - Simple Linux Utility for Resource Management
- ➢ It is a local manager that provides a framework for job queues, allocation of compute nodes, and the start and execution of jobs.

Local Resource Manager – SLURM - Components


Client commands (partial list)
scontrol
squeue
sbatch
srun
sacctmgr
sacct
Controller daemons
slurmctld
secondary slurmctld (optional)
slurmdbd (optional but recommended)
Other clusters (optional)
Database
slurmd
slurmd
slurmd
...
slurmd
Compute node daemons

Local Resource Manager – SLURM - Commands

The list and descriptions of the mostly used Slurm commands

● **sbatch** <script> To submit the job on HPC cluster

● **squeue** To see the status of all jobs submitted on the cluster
squeue -u <user name> To see status of user's jobs only. Also shows job-id.

●**sinfo** Provides the basic information about the resources on HPC cluster such as
- ➢Partitions/queue such as for cpu / gpu / high memory nodes
- ➢ Number of nodes for each type and their numbering/names
- ➢State of the nodes

Local Resource Manager – SLURM - Commands

- **scancel <job ID>** To delete the submitted jobs
- **scontrol show job <Job ID>** shows detailed information about a specific job or all jobs if no job id is given
- sacct –u <username> showing accounting stats after job ends
- **srun** To get resources in interactive mode
  - srun --nodes=1 --ntasks-per-node=1 --time=00:05:00 --pty bash

SLURM: Sample Job Script for Serial Jobs

```
#!/bin/bash
#SBATCH -J TestJob
#SBATCH -N 1                          // number of nodes
#SBATCH --ntasks-per-node=1            // number of cores per node
#SBATCH --output=3mm.out               // name of output file
#SBATCH --error=3mm.err                // name of error file
#SBATCH --partition=standard           // partition or queue name
#SBATCH –time=01:00:00                // time required to execute the program
```

SLURM: Sample Job Script for Parallel Jobs on CPU

```
#!/bin/bash
#SBATCH -N 1                                       // number of nodes
#SBATCH --ntasks-per-node=40      // number of cores per node
#SBATCH --output=3mm.out           // name of output file
#SBATCH --error=3mm.err              // name of error file
#SBATCH --partition=standard        // partition or queue name
#SBATCH –time=01:00:00               // time required to execute the program


export OMP_NUM_THREADS=40
```

SLURM: Job Script for Parallel Jobs on GPUs

```
#!/bin/bash
#SBATCH -N 1                              // number of nodes
#SBATCH --ntasks-per-node=40      // number of cores per node
#SBATCH --output=3mm.out            // name of output file
#SBATCH --error=3mm.err              // name of error file
#SBATCH –time=01:00:00              // time required to execute the program
#SBATCH --gres=gpu:2                  // request use of GPUs on compute nodes
#SBATCH --partition=gpu              // partition or queue name


export OMP_NUM_THREADS=40
```

PARAM Utkarsh

838TF

CPU Only Compute Nodes

- 107 Nodes
- 5136 Cores
- Compute power of Rpeak 476.6 TFLOPS
- Each Node with
  - 2 X Intel Xeon Cascadelake 8268, 24 cores, 2.9 GHz, processors
  - 192 GB memory
  - 480 GB SSD

GPU Compute Nodes

- 10 Nodes
- 400 CPU Cores
- 102400 CUDA Cores
- Rpeak CPU 32 TFLOPS + GPU 156 TF
- Each Node with
  - 2 X Intel Xeon Skylake 6248, 20 cores, 2.5 GHz, processors
  - 192 GB Memory
  - 2 x NVIDIA V100 SXM2 GPU Cards
  - 480 GB SSD

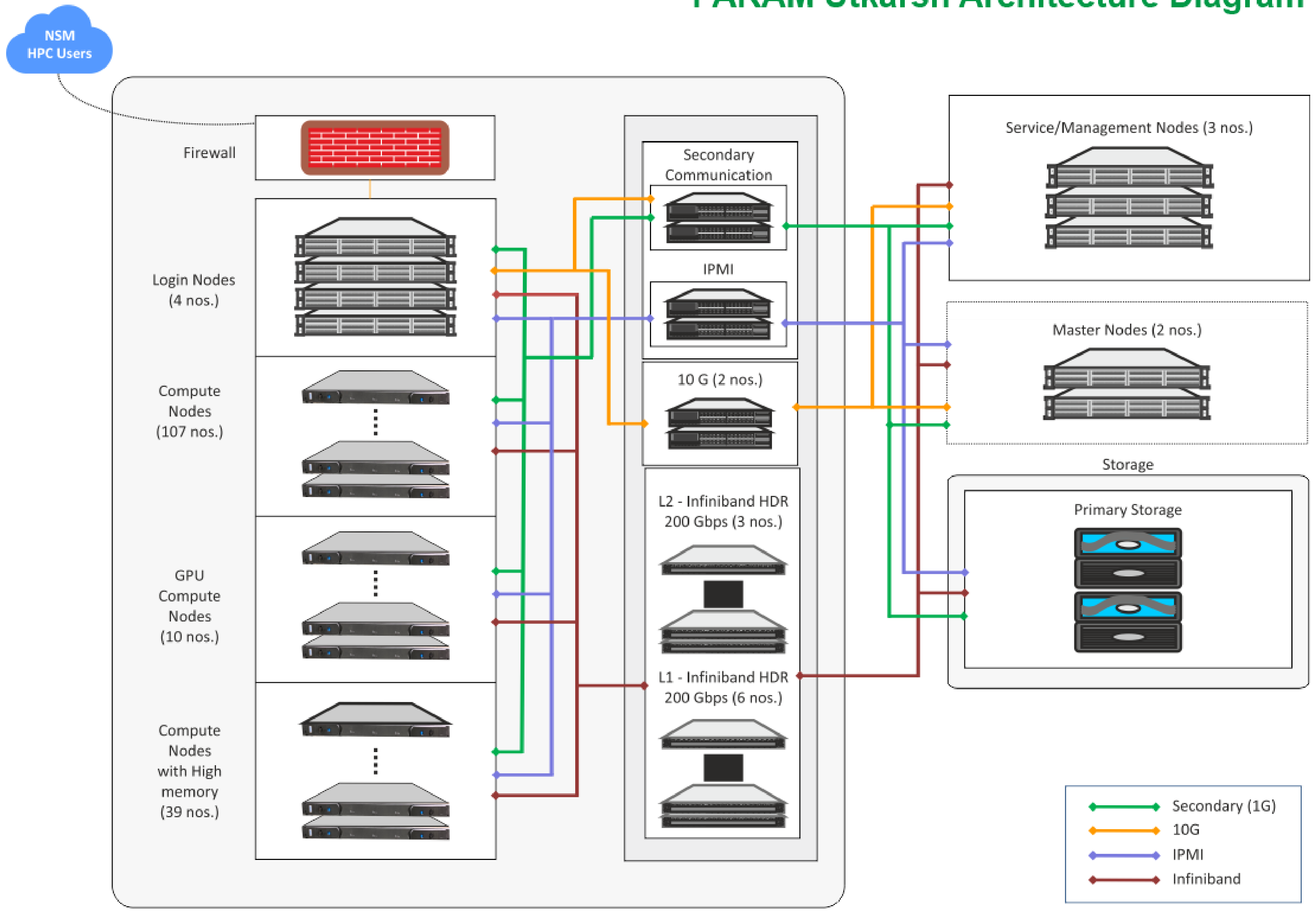
High Memory Compute Nodes

- 39 Nodes
- 1872 Cores
- Compute power of Rpeak 173.7 TFLOPS
- Each Node with
  - 2 X Intel Xeon Cascadelake 8268, 24 cores, 2.9 GHz, processors
  - 768 GB Memory
  - 480 GB SSD

System Details

Parameter	CPU only(75)	GPU Nodes(10)	GPU Ready(32)	HM Nodes(39)
Processor	2 x Xeon platinum 8268	2 x Xeon G-6248	2 x Xeon platinum 8268	2 x Xeon platinum 8268
Cores	48	40	48	48
Speed	2.9 GHz	2.5 GHz	2.9 GHz	2.9 GHz
Memory	192 GB	192 GB	192 GB	**768 GB**
HDD	480GB SSD	480GB SSD	480GB SSD	480GB SSD
Total cores	**3600**	**400**	**1536**	**1872**
Total Memory	14400 GB	1920 GB	6144 GB	29952 GB
	-	2 x NVIDIA V100	-	-

PARAM Utkarsh Architecture Diagram



Login - access methods

System Access

Accessing the cluster

- ➢ The cluster can be accessed through login nodes , which allows users to login.
- ➢ You may access login node through ssh.

Remote Access

a) Using SSH in Windows

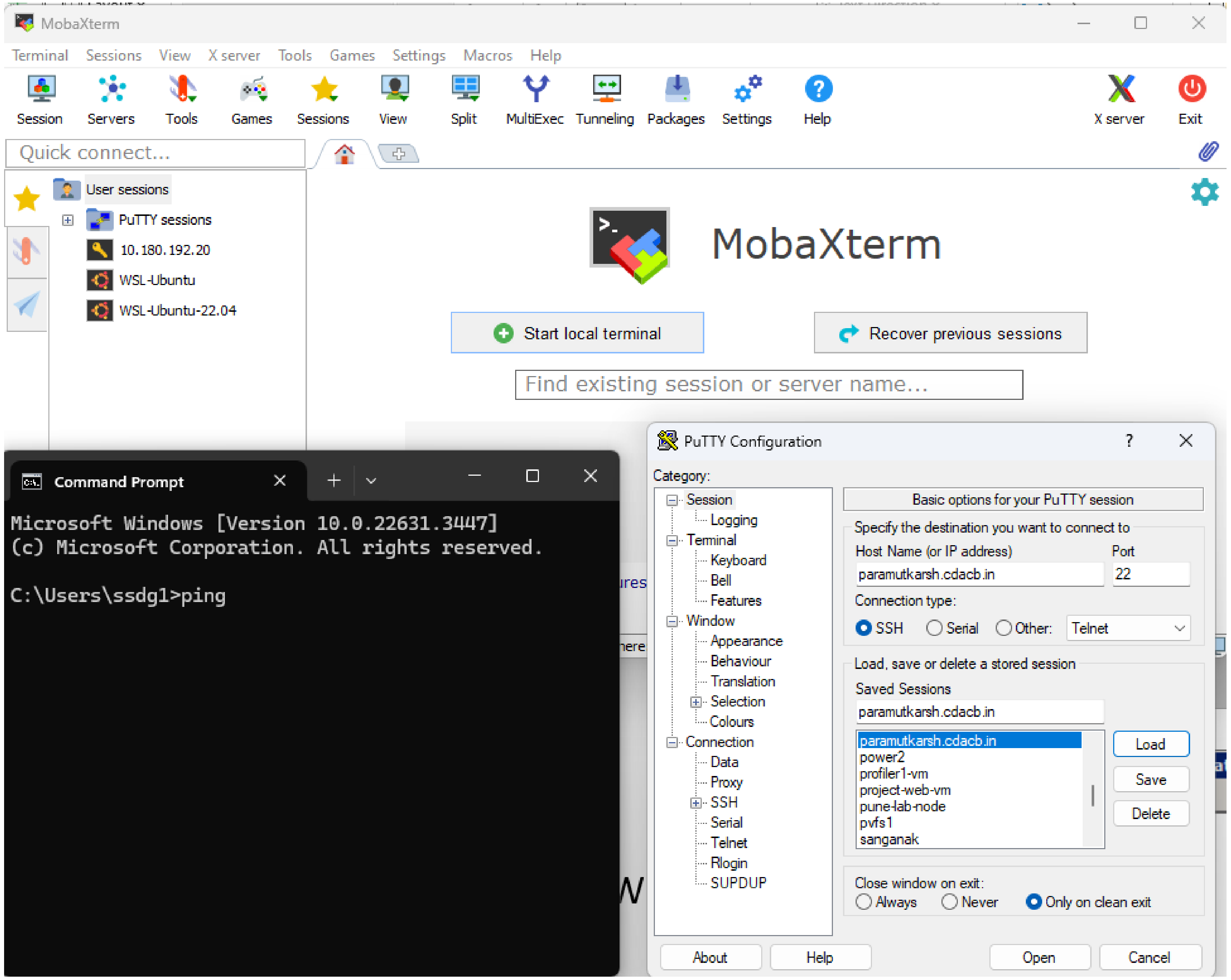
- ➢ To access PARAM Utkarsh you need to “ssh” the login server. PuTTY is the most popular open source “ssh” client application for Windows

b) Using SSH in Mac or Linux

- ➢ Both Mac and Linux systems provide a built-in SSH client, so there is no need to install any additional package. Open the terminal, connect to an SSH server by typing the following command:

```
ssh user1@paramutkarsh.cdacb.in
```

SSH Clients



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ssdg1>ping
```

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): paramutkarsh.cdacb.in   Port: 22

Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions: paramutkarsh.cdacb.in

paramutkarsh.cdacb.in
power2
profiler1-vm
project-web-vm
pune-lab-node
pvfs1
sanganak

Close window on exit: Always Never [✓ Only on clean exit]

Working Environment

- In Linux, on the Unix core utilities are in your command-PATH by default.
- In Linux, only the default system libraries are in your LD_LIBRARY_PATH
- The module system allows users to easily update their working environment, to include specific codes, versions, compilers, and libraries.

Module Utility

How to set Environment ?

- ➢ By default no application is set in your environment. User must explicitly set required ones
- ➢ **module** is the utility (also command name) to enable use of applications/ libraries / compilers available on the HPC cluster.
- ➢Module structure on Cluster

● apps/<application name>/version	:Applications available on the cluster
● compiler/<compiler name>/version	:Compilers available on the cluster
● lib/<library name>/version	:Available libraries
●	

Module Utility

➢Some Important commands:

- **module avail** To see the available software installed on HPC system
    - list of precompiled applications
    - different compilers and libraries (compilers include GNU, Intel, PGI)
- **module list** Shows the currently loaded modules in your shell
- **module load <Name of the module>**
    - *module load intel/2020*

      (to set Intel compilers version 2020 environment)
    - *module load   namd/intel2018/2.12*

      (to set NAMD app version 2.12 environment)

Module Utility

- **module unload <Name of the module>** This will remove all environment setting related to loaded previously
- **module purge** To clear all the loaded modules

Transferring files between local machine and HPC cluster

- ➢ When a user wishes to transfer data from their local system (laptop/desktop) to HPC system They can just use “scp” command on their terminal
- ➢ The command shown below can be used for effecting file transfers

  scp –r <path to the local data directory> <your username>@<IP of ParamUtkarsh>:<path to directory on HPC where to save the data>

  Example:

  scp *–r /dir/dir/file saurabh@<cluster IP/Name>:/home/Saurabh*
- ➢ Command could be used to transfer data from HPC system to your local system

  scp –r <path to directory on HPC> <your username>@<IP of local system>:<path to the local data directory>

  scp *–r /home/saurabh saurabh@<local system IP/Name>:/dir/dir/file*

SLURM Job Arrays

- Job arrays allow you to leverage SLURM’s ability to create multiple jobs from one script.
- For example, instead of having 5 submission scripts to run the same job with different arguments, you can have one script to run the 5 jobs at once.
- Many of the situations where this is useful include:
  - Running the same analysis program multiple times against different files or data sets.
  - Running the same program multiple times with different arguments.
  - Running a single program multiple times to analyzing a single data file.

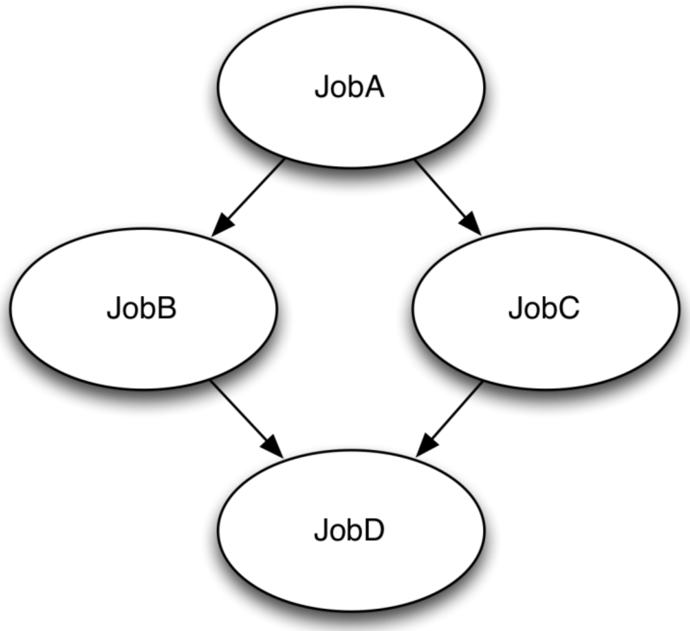
SLURM Job Arrays

- #SBATCH --array=*indexes*

1-10	1,2,3,4,5,6,7,8,9,10
2-20:2	2,4,6,8,10,12,14,16,18,20
1,3,5,7,11,21	1,3,5,7,11,21
2-20%2	2,4 then 6,8 then 10,12 …

Dependent Jobs/Workflows

- The job dependency feature of SLURM is useful when you need to run multiple jobs in a particular order.
- A standard example of this is a workflow in which the output from one job is used as the input to the next



Dependent Jobs Submission

- Job A

```
#!/bin/bash
#SBATCH --job-name=JobA
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --output=JobA.stdout
#SBATCH --error=JobA.stderr
echo "I'm job A"
echo "Sample job A output" > jobA.out
sleep 120
```

Dependent Jobs Submission

- Job B

```
#!/bin/bash
#SBATCH --job-name=JobB
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --output=JobB.stdout
#SBATCH --error=JobB.stderr
echo "I'm job B"
echo "I'm using output from job A"
cat jobA.out >> jobB.out
echo "" >> jobB.out
echo "Sample job B output" >> jobB.out
sleep 120
```

Dependent Jobs Submission

- Job C

```
#!/bin/bash
#SBATCH --job-name=JobC
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --output=JobC.stdout
#SBATCH --error=JobC.stderr
echo "I'm job C"
echo "I'm using output from job A"
cat jobA.out >> jobC.out
echo "" >> jobC.out
echo "Sample job C output" >> jobC.out
sleep 120
```

Dependent Jobs Submission

- Job D

```
#!/bin/bash
#SBATCH --job-name=JobD
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --output=JobD.stdout
#SBATCH --error=JobD.stderr
echo "I'm job D"
echo "I'm using output from jobs B and C"
cat jobB.out >> jobD.out
echo "" >> jobD.out
cat jobC.out >> jobD.out
echo "" >> jobD.out
echo "Sample job D output" >> jobD.out
sleep 120
```

Dependent Jobs Submission

- Submit Job A

```
[shamjith@login4 demo]$ sbatch JobA.slurm
Submitted batch job 666898
```

- Submit Jobs B and C

```
[shamjith@login4 demo]$ sbatch -d afterok:666898 JobB.slurm
Submitted batch job 666899
[shamjith@login4 demo]$ sbatch -d afterok:666898 JobC.slurm
Submitted batch job 666900
```

- Submit Job D

```
[shamjith@login4 demo]$ sbatch -d afterok:666899:666900 JobD.slurm
Submitted batch job 666901
```

Job Script - Best Practices

- Keep unique copies of the stdout and strderr

```
#SBATCH -o jobname.%j.o
#SBATCH -e jobname.%j.e
```

- echo commands back

```
#!/bin/bash -x
set -x
```

- print statements

```
input=file1.inp
echo $input
```

- print runtime environment

```
env
```

- make unique directories

```
mkdir -pv /home/$USER/${SLURM_JOB_ID}.${input}
```

Types of Errors - Overview

- Scheduler (SLURM)
- Syntax
- Memory
- Storage
- File access
- Network
- Parallel communication

Types of Errors - SLURM

- Scheduler (SLURM)
  - errors executing commands (sbatch, squeue)

```
sbatch: error: Batch job submission failed: Unable to contact slurm controller
```

```
squeue: error: slurm_receive_msg: Socket timed out on send/recv operation
slurm_load_jobs error: Socket timed out on send/recv operation
```

slurmctld process may be getting high volume of requests

Types of Errors - SLURM

- Scheduler (SLURM)
  - job states (ouput from squeue or sacct)

		State	Description
	CA	CANCELLED	Cancelled by the user or sysadm via scancel
3	CD	COMPLETED	All processes on all nodes completed with exit code = 0
	CG	COMPLETING	Job done, but processes on some nodes may still be active
	F	FAILED	Job terminated with non-zero exit code
	NF	NODE_FAIL	Job terminated due to failure of one of the allocated nodes
1	PD	PENDING	Job is awaiting resource allocation
	PR	PREEMPTED	Job terminated due to preemption
2	R	RUNNING	Job currently has an allocation
	S	SUSPENDED	Execution has been suspended and CPUs
	TO	TIMEOUT	Job terminated upon reaching its time limit (-t D-HH:MM)

Normal state changes are:

Types of Errors - Syntax

- Syntax
  - job script

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 1:00:00
#SBATCH --mem=4000
#SBATCH -partition general
```

```
sbatch: error: Invalid argument: general
```

```
# This is a Job Script for Syntax Errors
input file1.txt

echo $input
```

```
/var/slurmd/spool/slurmd/job70807187/slurm_script: line 8:    input: command not found
```

Types of Errors - Memory

- Memory
  - out of memory

```
slurmstepd: error: Exceeded step memory limit at some point.
```

  - malloc failure
    - C function that allocates bytes of memory and returns a pointer to the allocated memory
  - SIGSEGV, segfault or segmentation violation
    - arise primarily due to errors in use of pointers for virtual memory addressing, particularly illegal access.

Types of Error - Storage

- Storage
  - out of space on device

```
cp: closing `pgm_simulation.out': No space left on device
```

  - out of space on filesystem quota
  - out of inodes / file descriptors

```
cp: cannot create regular file `simulation.sh': Disk quota exceeded
```

Types of Errors – File Access

```
# This is a Job Script for Syntax Errors
input=/n/home_rc/pedmon/a.out

cat $input
mpirun a.out
```

- File access
  - no permission to read/write

```
/home/user/a.out: Permission denied.
```

  - file or library not found

```
/home/user/a.out: error while loading shared libraries: libm1.so.0: cannot open shared object
file: No such file or directory
```

  - command not found

```
/var/slurmd/spool/slurmd/job2255/slurm_script: line 16: mpirun:            command not found
```

Performance Analysis

Speedup

- Speedup measures change in running time due to parallelism. The number of PEs is denoted here by n.
- Based on running times, $S(n) = t_s/t_p$, where
  - t_s is the execution time on a single processor, using the fastest known sequential algorithm
  - t_p is the execution time using a parallel processor.

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

Amdahl’s Law

This law, formulated by computer architect Gene Amdahl, quantifies the maximum potential speedup achieved by parallelizing a computation while considering the fixed portion of the task that cannot be parallelized.

Let *f* be the fraction of operations in a computation that must be performed sequentially, where $0 \le f \le 1$. The maximum speedup ψ achievable by a parallel computer with *n* processors is

$$\psi \equiv S(n) \le \frac{1}{f + (1 - f)/n} \le \frac{1}{f}$$

Example 1

- 95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$\psi \leq \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

Gustafson’s Law

- Gustafson's Law, formulated by computer scientist John L. Gustafson, provides an alternative perspective on the potential performance improvements achievable through parallel computing.
- Gustafson's Law considers the workload's scalability and disregards the idea of having a fixed portion of the workload.
- It states that as the computing resources (typically the number of processors) increase, the problem size and workload can be scaled up to utilize the available resources effectively.
- This shows that even more significant problems can be solved with parallel computing, which is more efficient as it focuses on keeping the problem size proportional to the available resources.
- $$S(n)=n+(1-n)f$$

$S(n)$ is the theoretical speedup with n processors, and f is the serial fraction of the job

Granularity

- For parallel tasks we talk about the *granularity* – size of the computation between *synchronization points*
  - Coarse – heavyweight processes + IPC (interprocess communication,like MPI)
  - Fine – Instruction level (eg. SIMD)
  - Medium – Threads + [message passing + shared memory]
- Computation to Communication Ratio
  - (Computation time)/(Communication time)
  - *Increasing* this ratio is often a key to good efficiency

Communication Overhead

- Another important metric is *communication*
  - Measure time spent on communication that *cannot* be spent on computation
- *Overlapped Messages* –
  - portion of message lifetime that can occur concurrently with computation

Debugging

Parallel vs. serial

- Parallel programming is more difficult than serial programming each step of the way:
  - Designing stage
  - Coding
  - Debugging
  - Profiling
  - Maintenance

Parallel bugs

- In addition to usual, “serial” bugs, parallel programs can have “parallel-only” bugs, such as
  - Race conditions
    - when two or more threads access shared data concurrently, and the final outcome depends on the order of execution.
  - Deadlocks
    - when two or more tasks or threads are blocked, each waiting for the other to release a resource or respond to a signal.

Race condition

- In OpenMP, race conditions result from misuse of shared variables, when
    - A variable is mistakenly labeled as shared (where in fact it needs to be private), or
    - A variable is correctly labeled as shared, but the access to the variable wasn't properly protected (serialized)
    - For example, if two threads execute the code concurrently, one thread might read the value of b while another thread is updating it, leading to inconsistent results
        - a = b + c;
        - b = a + c;
        - c = b + a;

Race condition

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
   int numt, tid;
   #pragma omp parallel
   {
      // All threads are accessing and modifying the shared variables numt and tid
      numt = omp_get_num_threads();
      tid = omp_get_thread_num();
      printf("Hello world from thread %d of %d\n", tid, numt);
   }
   return 0;
}
```

```
Hello world from thread 1 of 4 Hello
world from thread 0 of 4 Hello world
from thread 1 of 4 Hello world from
thread 2 of 4
```

Race condition-Solution

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
  int numt, tid;

  #pragma omp parallel private(tid) shared(numt)
  {
    numt = omp_get_num_threads();
    tid = omp_get_thread_num();

    printf("Hello world from thread %d of %d\n", tid, numt);
  }

  return 0;
}
```

```
Hello world from thread 1
of 4 Hello world from
thread 0 of 4 Hello world
from thread 3 of 4 Hello
world from thread 2 of 4
```

Example of race condition

Shared variable incremented by multiple threads without protection.

```
…
int main() {
   int shared_var = 0;
   #pragma omp parallel num_threads(4)
   {
      #pragma omp for
      for (int i = 0; i < 1000; i++) {
         shared_var++;
      }
   }
   printf("Shared variable value: %d\n", shared_var);
   return 0;
}
```

Use synchronization constructs (e.g., critical sections, locks, atomic operations) to protect access to shared variables

```
…
int main() {
   int shared_var = 0;
   #pragma omp parallel num_threads(4)
   {
      #pragma omp for
      for (int i = 0; i < 1000; i++) {
         #pragma omp critical
         {
            shared_var++;
         }
      }
```

Deadlocks

- It happens when thread(s) lock up while waiting on a locked resource that will never become available
- The sign of a deadlock: the program hangs (always or sometimes) when reaching a certain point in the code

Deadlocks (cont.)

- Prevention strategies:
    - Be very careful with conditional clauses using threadID as an argument, as common OpenMP constructs (for/do, single) require all the threads in the team reaching them.
    - Communications between threads (using a shared variable) have to use “flush” pragma, on both writing and reading sides. – Don't forget to unset locks after setting them.

Example of a deadlock

we can ensure that the locks are always acquired and released in the same order in all sections.

```
if (thread_id == 0) {
    omp_set_lock(&locka);
    printf("Thread 0 locked locka\n");
    omp_set_lock(&lockb);
    printf("Thread 0 locked lockb\n");
    omp_unset_lock(&lock1);
    omp_unset_lock(&lock2);
} else {
    omp_set_lock(&lockb);
    printf("Thread 1 locked lockb\n");
    omp_set_lock(&locka);
    printf("Thread 1 locked locka\n");
    omp_unset_lock(&lock1);
    omp_unset_lock(&lock2);
}
```

```
Thread 1                Thread 2
-------------------      -------------------
set Lock A            |   set Lock B
wait for Lock B       |   wait for Lock
A
```

(deadlock)

Overview of GDB

- "GNU Debugger"
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like segmentation faults may be easier to find with the help of gdb
- http://sourceware.org/gdb/current/onlinedocs/gdb toc.html - online manual

Preparing your program

- Additional step when compiling program
- Add a –g option to enable built-in debugging support (which gdb needs)
- –g which tells the compiler to generate symbolic information required by any debugger.
- For examples
- gcc –g –o0 –fopenmp –o code code.c

GDB Commands

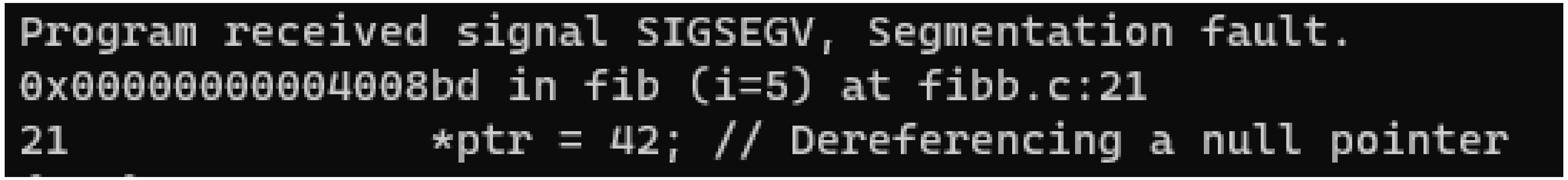
- Related Gdb Commands:
  - List: list the source code and each execution’s corresponding line number
  - Break linenumber: set breakpoint at the linenumber
  - Run argv: run the execution code with the parameter argv
  - Next: execute the next line of code
  - Backtrace: show trace of all function calls in stack
  - **Info frame**: List address, language, address of arguments/local variables and which registers were saved in frame.
    - This will show where the return address is saved
    - Return address is in Register EIP
    - Calling stack pointer is in Register EBP
  - x &variable: show the address and value of a local variable (in hex format)
  - **x address**: print binary representation of bytes of memory pointed to by address.

Example of Using GDB - Serial

```
#include <stdio.h>
void foo(char * input){
    int a1=11;
     int a2=22;
     char buf[7];
    strcpy(buf, input);
}
void main(int argc, char **argv){
   foo(argv[1]);
}
```

Starting up gdb

- Just try “gdb” or “gdb program.x” to access the GDB prompt
  - *(gdb)*
- To execute the program
  - *(gdb) run*
- If the program encounters issues, GDB provides valuable information.
- For example: Segmentation Fault



```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004008bd in fib (i=5) at fibb.c:21
21              *ptr = 42; // Dereferencing a null pointer
```

Setting breakpoints

- Breakpoints can be used to stop the program run in the middle, at a designated point.
- Useful for investigating issues or examining specific section of code
- Setting Breakpoints - File and Line:

  Syntax: (gdb) break file.c:6

  Set a breakpoint at line 6 in the file "file.c."
- Setting Breakpoints - Function:

  Syntax: (gdb) break my_func

  Set a breakpoint at the beginning of the function "my_func."

Now what ?

- After setting a breakpoint, use the *run* command to execute the program until it reaches the specified breakpoint.
- Stepping to the Next line
  - $(gdb) next
- Stepping into functions
  - $(gdb) step
- Listing Source code
  - $(gdb) list

Other useful commands

- Inspect variable values during the paused state.
  - $(gdb) print <variable>
- Continue execution after inspection.
  - $(gdb) continue
- Examining Stack Status:
  - $(gdb) backtrace

OpenMP Debugging with GDB

- Introduction to GDB as a Debugger for OpenMP Programs:
    - GDB (GNU Debugger) is a powerful command-line debugger for C/C++ programs on UNIX systems.
    - Widely used for debugging OpenMP programs to identify and resolve issues.
- Commands and Techniques for Debugging OpenMP Code with GDB:
    - Start up GDB to debug a program
-
```
$ OMP_NUM_THREADS=2 gdb ./application.exe
```

OpenMP Debugging with GDB

Thread-specific Commands in GDB

- (gdb) info thread- Prints out information about all current threads.
  - Helpful for understanding the status of threads during program execution, especially in parallel sections.
- (gdb) thread -Prints the current thread
- (gdb) thread <thread_no> - switches to specific thread.
  - Allows the user to switch between threads and focus debugging efforts on specific threads of interest.

OpenMP Debugging with GDB

- thread apply- Sending GDB Commands to Multiple Threads
    - Useful for applying debugger commands uniformly across multiple threads.
    - (gdb) thread apply 2-4 continue
    - (gdb) thread apply all print tid
- Scheduling Locking:
    - (gdb) set scheduler-locking on
    - (gdb) set scheduler-locking off
    - (gdb) show scheduler-locking

Debuggers

- GDB
- LLDB
- TotalView
- DDT

Profiling

What is Profiling

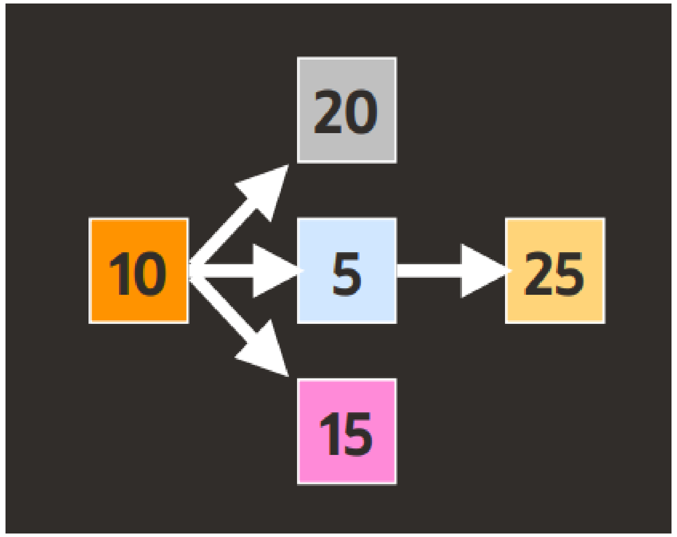
- Application profiling is the process of analysing and measuring the performance of a software application in order to identify and diagnose performance issues or bottlenecks.
- Profiling tools help developers to identify areas of code that are taking a long time to execute, or that are using an excessive amount of system resources like memory, CPU or disk I/O.
- The purpose of profiling is to identify any bottlenecks or areas of the application that can be optimized to improve its performance, scalability, and reliability.
- By profiling their applications, developers can ensure that they are delivering high-quality software that performs well and meets the needs of their users.
- Profiling involves collecting data on various aspects of the application and its analysis.
- Implemented through either
  - Sampling based profiling
  - Instrumentation based Profiling

Sampling Based Profiling

- With sampling, the executable is stopped at regular intervals. Each time it is halted, key information is gathered and stored.
- At specified intervals, the Sampling method collects information about the functions that are executing in your application. While the program is interrupted the profiler grabs a snapshot of its current state.
- Sampling is a low cost to the profiler and has little effect on the execution of the application being profiled.
- If you need accurate measurements of call times or are looking for performance issues in an application , then sampling based profiler are useful.
- Sampling has less accuracy in the number of calls

Inclusive Vs Exclusive CPU Time

- This is an important concept in profiling tools
- The inclusive metric includes all callees underneath the caller - For example, all the CPU time accumulated when executing a function
- The exclusive metric excludes everything outside the caller - For example, the CPU time accumulated outside of calling other functions



Function	Inclusive time	Exclusive time
A	75	10
B	20	20
C	30	5
D	15	15
E	25	25

```
Void Alpha()
{
                    30 samples
 Beta();

}

Void Beta()
{
                    50 samples

}
```

Functions	Inclusive	Exclusive
Alpha	80	30
Beta	50	50

Instrumentation based Profiling

- The first and earliest type are instrumenting profilers.
- Data collection is done by tools that either injecting code into a binary file that captures timing information or by using callback hooks to collect and emit exact timing and call count information while an application runs.
- The instrumentation method has a high overhead when compared to sampling-based approaches.
- Instrumentation profiling is that you can get exact call counts on how many times your functions were called.
- Instrumentation can be achieved during
  - Compile time Instrumentation
  - Binary Instrumentation

Performance Profilers

- TAU
- Vtune
- GProfNG
- HPCToolkit
- Likwid

Petascale and Exascale Computing

AIRAWAT - PARAM Siddhi AI

AIRAWAT - PARAM Siddhi - AI of 8.5 Petaflops is the fastest Supercomputer in India and ranked at No. 90 position in ‘TOP500 Supercomputer List – November 2023’

List of systems commissioned in Phase-I and Phase-II (24PF)

S.No.	Institute Name	HPC System Name	Computing Power	Year of Commissioning
1.	IIT(BHU), Varanasi	PARAM Shivay	838TF	2019
2.	IISER, Pune	PARAM Brahma	1.7PF	2020
3.	IIT, Kharagpur	PARAM Shakti	1.66PF	2020
4.	JNCASR, Bangalore	PARAM Yukti	1.8PF	2020
5.	IIT, Kanpur	PARAM Sanganak	1.66PF	2020
6.	C-DAC, Pune (National AI Facility)	PARAM Siddhi	5.2PF/210PF (AI)	2020
7.	IIT, Hyderabad	PARAM Seva	838TF	2021
8.	NABI, Mohali	PARAM Smriti	838TF	2021
9.	IISc, Bangalore	PARAM Pravega	3.3PF	2022
10.	C-DAC, Bangalore (MSME)	PARAM Utkarsh	838TF	2021
11.	IIT, Roorkee	PARAM Ganga	1.66PF	2022
12.	IIT, Gandhinagar	PARAM Ananta	838TF	2022
13.	NIT, Trichy	PARAM Porul	838TF	2022
14.	IIT, Guwahati	PARAM Kamrupa	838TF	2022
15.	IIT, Mandi	PARAM Himalaya	838TF	2022

NSM Commissioned Sites

PARAM Shivay

PARAM Shakti

PARAM Brahma

PARAM Yukti

PARAM Sanganak

PARAM Pravega

PARAM Seva

PARAM Smriti

PARAM Himalaya

PARAM Kamrupa

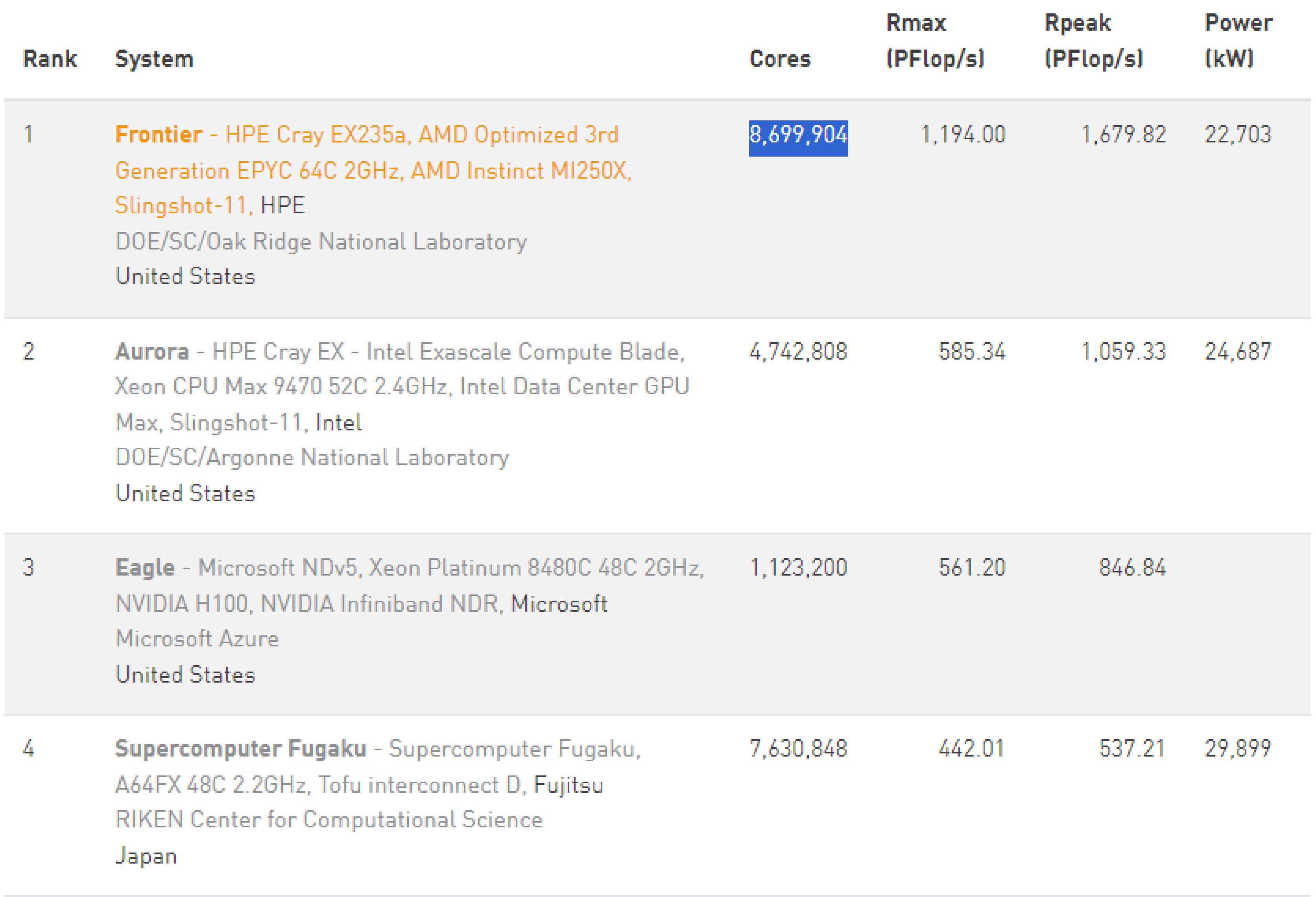
PARAM Utkarsh

PARAM Ganga

PARAM Ananta

PARAM Porul

Top 500 Supercomputers



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	4,742,808	585.34	1,059.33	24,687
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States	1,123,200	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899

Frontier Exascale Supercomputer

1.1EF

700
PETABYTES

FRONTIER
OAK RIDGE National Laboratory
U.S. DEPARTMENT OF ENERGY
Hewlett Packard Enterprise
AMD

40
MEGAWATTS

8699904
Cores

Challenges

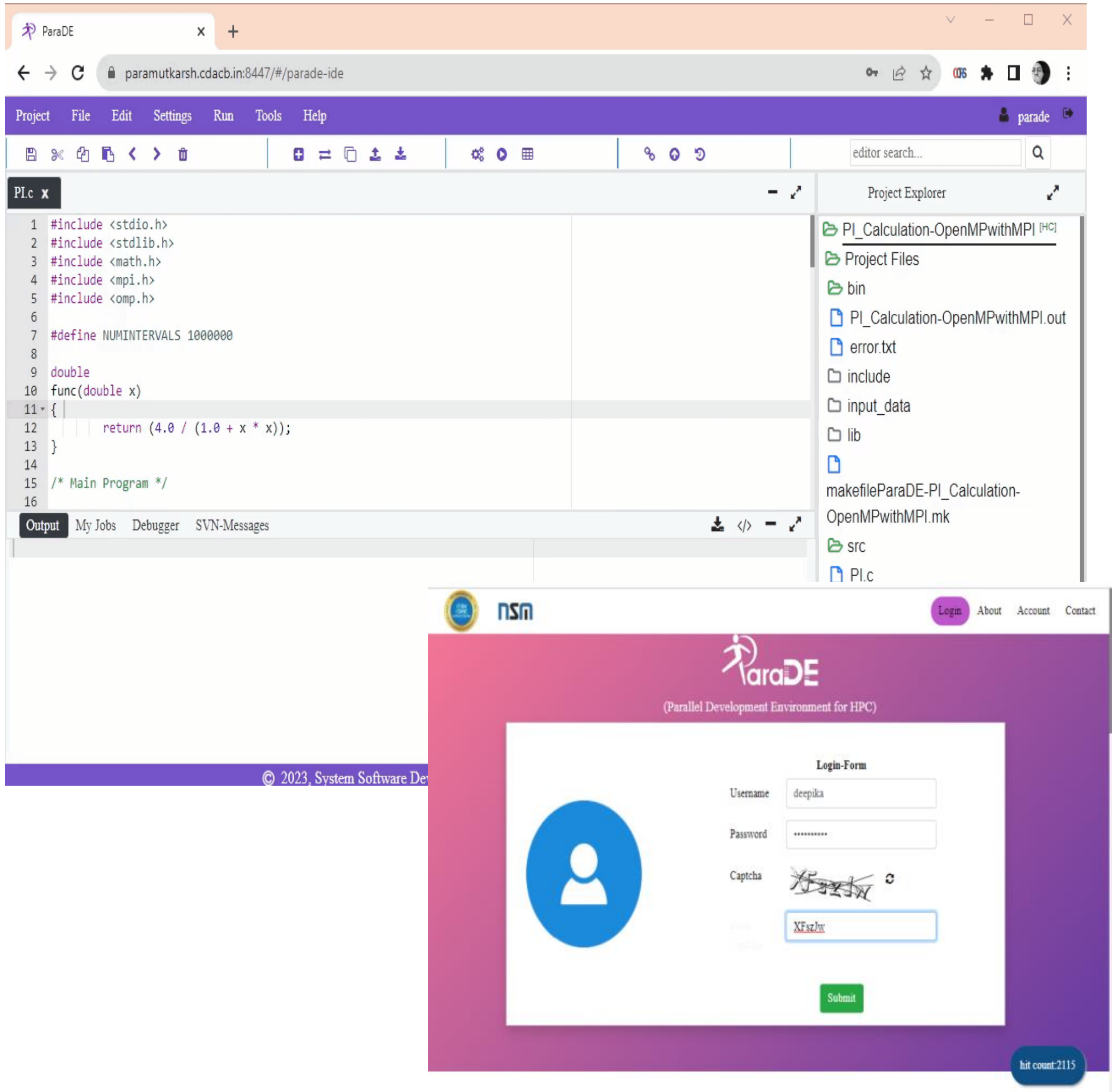
- Power consumption. ...
- Scalability. ...
- Heterogeneity
- Resilience. ...
- Programming methodologies and applications

C-DAC’s System Software for HPC

ParaDE (<u>Para</u>llel <u>D</u>evelopment <u>E</u>nvironment)

- It is an **IDE for HPC** which facilitates application development on HPC platform from anywhere and anytime

- Major Features
  - ✓ Support **multiple paradigms** like MPI, OpenMP, CUDA, OpenACC under a single project
  - ✓ **Automatic compilation**
  - ✓ **Assisted job submission**
  - ✓ Provision to **add custom** compilers, libraries and tools
  - ✓ **Version Control System**
  - ✓ Access to development tools (**debuggers, profilers, converters**) available on the platform

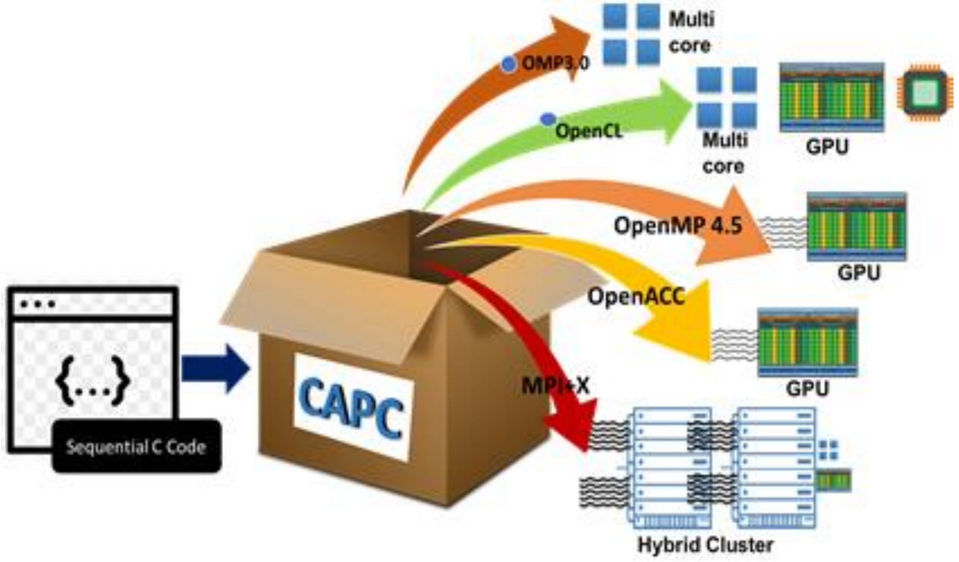


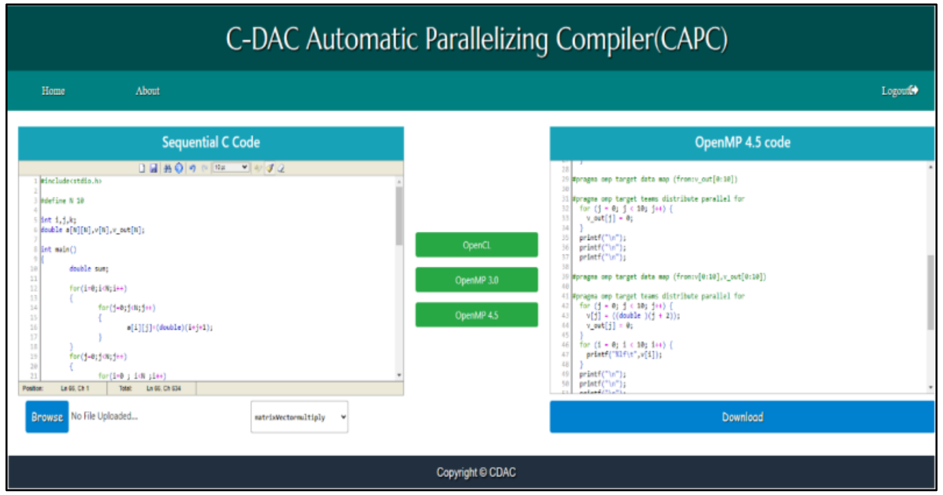
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#include <omp.h>

#define NUMINTERVALS 1000000

double
func(double x)
{
        return (4.0 / (1.0 + x * x));
}

/* Main Program */
```

Username: deepika
Password: ••••••••••
Captcha: XFszJr

CAPC - CDAC Automatic Parallelizing Compiler

- Automatically converts sequential programs to equivalent parallel programs for the target parallel architectures





```
#include<stdio.h>

#define N 10

int i,j,k;
double a[N][N],v[N],v_out[N];

int main()
{
        double sum;

        for(i=0;i<N;i++)
        {
                for(j=0;j<N;j++)
                {
                        a[i][j]=(double)(i+j+1);
                }
        }
        for(j=0;j<N;j++)
        {
                for(i=0 ; i<N ;i++)
```

Position: Ln 66, Ch 1   Total: Ln 66, Ch 634

```

#pragma omp target data map (from:v_out[0:10])

#pragma omp target teams distribute parallel for
  for (j = 0; j < 10; j++) {
    v_out[j] = 0;
  }
  printf("\n");
  printf("\n");
  printf("\n");

#pragma omp target data map (from:v[0:10],v_out[0:10])

#pragma omp target teams distribute parallel for
  for (j = 0; j < 10; j++) {
    v[j] = ((double )(j + 2));
    v_out[j] = 0;
  }
  for (i = 0; i < 10; i++) {
    printf("%lf\t",v[i]);
  }
  printf("\n");
  printf("\n");
```

❑Features

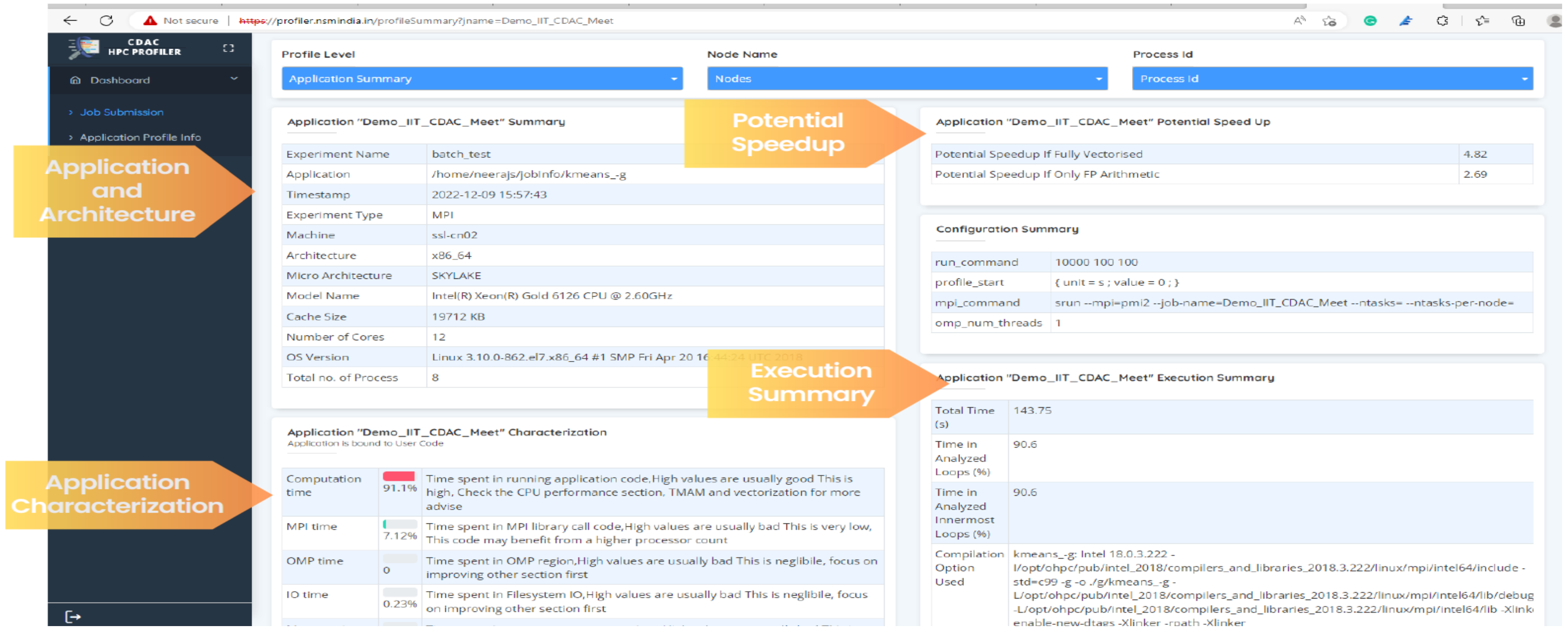
- Automatic parallelizes the code without user hints/inputs
- Support for multiple Parallel paradigms
- Human readable output
- Profitability estimate before parallelizing

CHAP - CDAC HPC Application Profiler

- It provide insights on the application performance behaviour at the cluster level and process level

- Major Features

  ✓ Application **Performance Summary**
  ✓ **Guided** profiling
  ✓ **Hotspot** identification
  ✓ Multi-dimension analysis
  ✓ Minimal overhead in profiling
  ✓ Potential **Performance suggestions**
  ✓ **Integration with LRMs**
  ✓ Access via **web interface**



Not secure | https://profiler.nsmindia.in/profileSummary?jname=Demo_IIT_CDAC_Meet
CDAC HPC PROFILER
Dashboard
Job Submission
Application Profile Info
Application and Architecture
Potential Speedup
Execution Summary
Application Characterization

Profile Level: Application Summary
Node Name: Nodes
Process Id: Process Id

Application "Demo_IIT_CDAC_Meet" Summary

Experiment Name	batch_test
Application	/home/neerajs/jobInfo/kmeans_-g
Timestamp	2022-12-09 15:57:43
Experiment Type	MPI
Machine	ssl-cn02
Architecture	x86_64
Micro Architecture	SKYLAKE
Model Name	Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
Cache Size	19712 KB
Number of Cores	12
OS Version	Linux 3.10.0-862.el7.x86_64 #1 SMP Fri Apr 20 16:44:24 UTC 2018
Total no. of Process	8

Application "Demo_IIT_CDAC_Meet" Potential Speed Up

Potential Speedup If Fully Vectorised	4.82
Potential Speedup If Only FP Arithmetic	2.69

Configuration Summary

run_command	10000 100 100
profile_start	{ unit = s ; value = 0 ; }
mpi_command	srun --mpi=pmi2 --job-name=Demo_IIT_CDAC_Meet --ntasks= --ntasks-per-node=
omp_num_threads	1

Application "Demo_IIT_CDAC_Meet" Characterization
Application is bound to User Code

Computation time	91.1%	Time spent in running application code,High values are usually good This is high, Check the CPU performance section, TMAM and vectorization for more advise
MPI time	7.12%	Time spent in MPI library call code,High values are usually bad This is very low, This code may benefit from a higher processor count
OMP time	0	Time spent in OMP region,High values are usually bad This is neglibile, focus on improving other section first
IO time	0.23%	Time spent in Filesystem IO,High values are usually bad This is neglibile, focus on improving other section first

Application "Demo_IIT_CDAC_Meet" Execution Summary

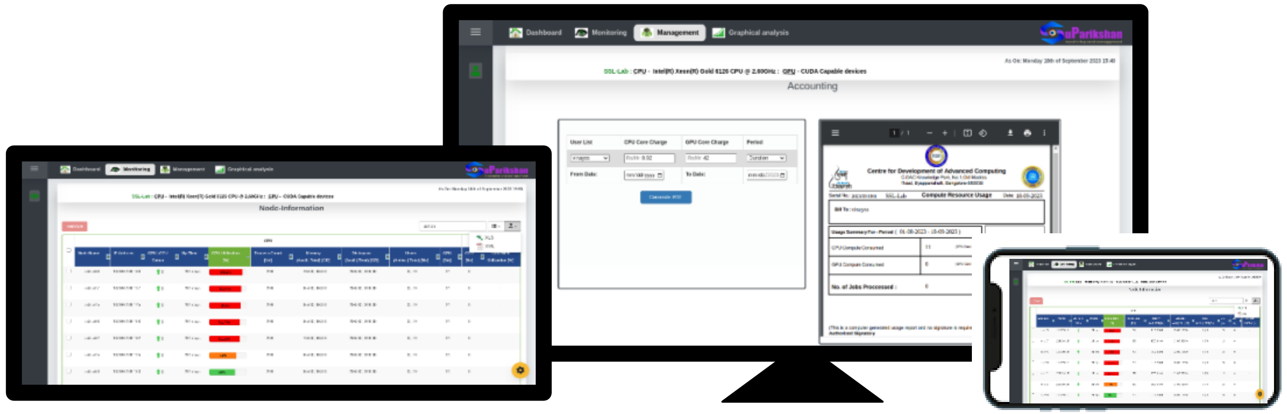
Total Time (s)	143.75
Time in Analyzed Loops (%)	90.6
Time in Analyzed Innermost Loops (%)	90.6
Compilation Option Used	kmeans_-g: Intel 18.0.3.222 -I/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/include -std=c99 -g -o ./g/kmeans_-g -L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/lib/debug -L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/lib -Xlink enable-new-dtags -Xlinker -rpath -Xlinker

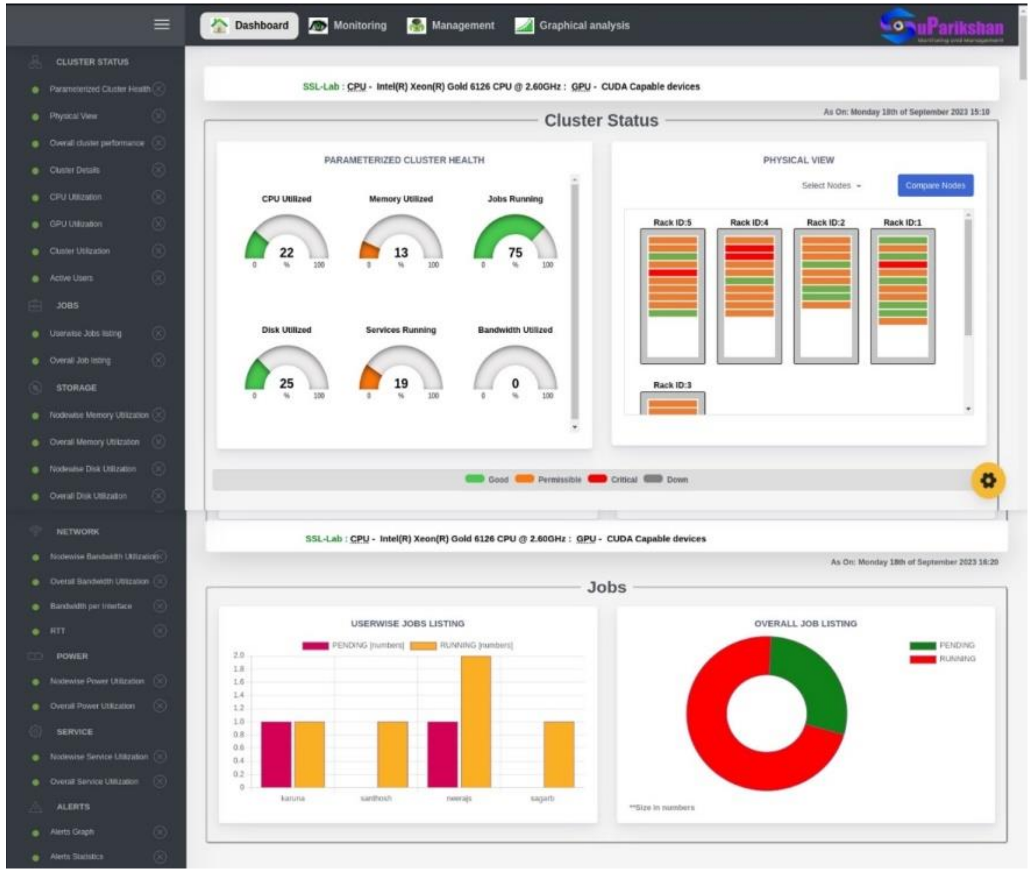
SuParikshan

- Monitoring & management solution for heterogeneous HPC cluster. It is a pluggable, customizable and integrate with other third party tools

Major Features

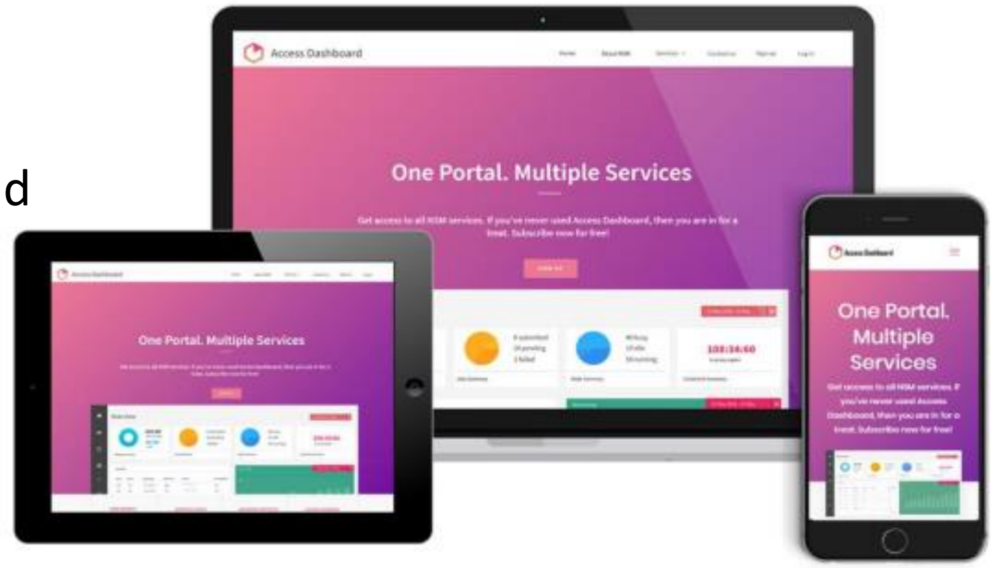
- ✓ Assist Administrators to manage heterogeneous infrastructure like CPU, GPU, FPGA etc
- ✓ Low Footprint software with responsive design
- ✓ Automatic error identification & ticketing system
- ✓ Supports monitoring customized scripts of admin
- ✓ Accounting of cluster resources and users





HPC Dashboard

HPC Dashboard hides the complexities of high performance computing (HPC). It has a rich, easy to use interface for end-users and administrators, designed to increase productivity through its visual web-interface, powerful job submission and management feature, remote visualization, accounting & reporting, secure web console, file manager, administration panel and other workload functions.


Access Dashboard
One Portal. Multiple Services
One Portal. Multiple Services
One Portal. Multiple Services

Features of HPC Dashboard

- Personalized Dashboard
- Secured Web Console
- Mad Libs Job submission
- Accounting & Reporting
- Advanced File manager
- Interactive Voice
- Integrated Help Desk
- Single Sign-On
- Resource Administration
- Remote Visualization

Questions