Matlab® Distributed Computing Server (MDCS) @ CBI Laboratory

Release: R2012a
Overview

1) Matlab Distributed Computing Server Introduction
2) Hardware/Software/Utilization @ CBI
3) Benefits of using the MDCS:
   Performance & License Utilization Minimization
4) Key Usage Scenarios
5) MDCS utilization examples
6) Hands-on Training
MDCS Introduction

- **3 key pieces:**
  - 1) Matlab client + Parallel Compute Toolbox
  - 2) Job Manager/Scheduler (Matlab Job Manager, LSF, PBS, Sun Grid Engine)
    - We currently use the Matlab Job Manager.
  - 3) MDCS Worker Processes (a.k.a. “Labs”)
    - The workers never request regular Matlab or toolbox licenses.
    - The only license an MDCS worker ever uses is an MDCS worker license.
    - Toolboxes are unlocked to an MDCS worker based on the licenses owned by the client during the job submission process.
The Matlab Distributed Computing Server is a cluster software infrastructure built over the Message Passing Interface to allow the scaling of Parallel Compute Toolbox enabled codes.

**Performance:** Provides compute+memory scalability

**Licensing:** Workers only need their MDCS worker license (64 @ CBI Lab), leaving regular Matlab and Toolbox licenses available for others to use (e.g. Statistics Toolbox).

A regular Matlab license + Parallel Compute Toolbox license is only needed during the job submission process.

Parallel Compute Toolbox constructs (parfor~OpenMP, spmd~MPI, spmd codistributed arrays~MPI) scale seamlessly from a local system to the Distributed Server.

Code development can be done on a user's workstation, then, when ready, the MDCS can be used to scale that code in both memory and compute dimensions.

Multiple levels of parallelism can be implemented using the MDCS:

- Independent jobs (Distributed jobs~Task Computing)
- Complex fully parallel algorithms that require inter-process communication and synchronization (Parallel jobs~ parfor, spmd, labSend, labReceive, spmd Co-distributed arrays)
MDCS Introduction

Non-Interactive Job Submission

Client
Matlab+
PCT+
Toolboxes

job=createJob()

submit()

job.fetchOutputs()

MDCS Scheduler
Pending
Queued
Running
Finished

MDCS Workers
Server#1
Labs 1-16

Server#2
Labs 33-48

Worker Licenses Only

Server#3
Labs 17-32

Server#4
Labs 49-64

License Manager
64 Worker Licenses

Cheetah Cluster
Compute Node
qlogin access

Ethernet
Hardware/Software/Utilization @ CBI

- MDCS worker processes run on 4 physical servers
- PowerEdge M910: Four x 16 core systems, 4x64GB RAM, 2x Intel Xeon 2.26 Ghz/system with 8 cores per processor
- Total of 64 cores, with 256 GB total RAM distributed among systems
- Max 64 MDCS worker licenses available
- Subsets of MDCS workers can be created based on project needs
- During a software consulting session we can evaluate your code & project requirements and assist you in transforming your code to a form that will be runnable on the MDCS cluster.
  - We'll see more of the changes to code needed to run on the MDCS cluster in the hands-on examples.
  - Examples will take you through the process of moving a Matlab script to a function that can be submitted to the MDCS scheduler as a Job.
Benefits of MDCS

- **Performance: Scaling in compute & memory**
  - Running PCT code on MDCS Profile vs Local Profile
  - On a local profile, limit of 12 labs (R2012a) + Memory limits, IPC limits
  - Up to 64 labs on CBI MDCS cluster (limited by MDCS worker licenses)

- **Minimize regular Matlab + Toolbox license utilization (e.g. Statistics toolbox)**
  - Each MDCS worker uses only a single MDCS worker license
  - No regular Matlab licenses or toolbox licenses are checked out by MDCS workers

- **Running code requiring non-compilable toolboxes**
  - (SimBiology, others) without using up licenses
  - Job queues allow scaling to large number of jobs
  - For example, running many jobs for a parameter scan of a time consuming parallel enabled simulation.
  - Submit the jobs and the MDCS scheduler will manage the rest.
Benefits of MDCS

- **Rapid prototyping of parallel algorithms**
  - Using Matlab+PCT+MDCS instead of C/C++/Fortran+OpenMP+MPI directly
  - Memory scaling w/ co-distributed arrays
    - Minimize single-node memory utilization
      - Can enable processing larger datasets in a distributed manner.

- **Many built-in algorithms & toolboxes have some Parallel Compute Toolbox enablement**
  - e.g. fft, lu, svd, many more with SPMD(co-distributed arrays)
Key usage scenarios

1) Standalone Executable+MCR+Cheetah Cluster (Compilable Fully Independent Jobs)
   - If running code where each job is completely independent of other jobs, and your code is deployable with the Matlab Compiler Toolbox+Matlab Compiler Runtime, then the preferred approach is to run your deployed Matlab executable on the cheetah cluster as normal grid jobs.
   - Solves both the performance scaling problem and the license utilization problem.
   - You can submit many deployed executable based jobs to the cheetah cluster without using up any licenses.
   - One of our users was able to bring down project simulation runtime from months to days using this approach; enabling his Ph. D. Research. CBI provided both hardware + software improvements over his prior running environment.

2) MDCS (Non-compilable or Inter-process communication/ synchronization requiring jobs)
   - If approach 1 does not fit your project, either of 3 scenarios:
     - 2.1) Your jobs need to communicate with each other while they run concurrently.
     - 2.2) Your code does not deploy with the Matlab Compiler Toolbox (e.g. SimBiology)
     - 2.3) Your code already uses PCT toolbox functionality and you need to scale to more workers than are provided when running local (Max 12 labs as of R2012a in local profile). If items 2.1 and 2.2 do not apply to your code, it should be technically feasible to reorganize your code to fit approach 1 many times.
Key usage scenarios: Modes

- **Interactive Use:** (matlabpool/spmd/pmode/mpiprofile)
  - Should use only on a local system (e.g. one of the Workstations @ CBI) as part of initial algorithm development, but never on the MDCS cluster or on the cheetah cluster nodes (MDCS clients).
  - Use of interactive access to MDCS cluster bypasses scheduler interface leaving the system potentially locked for other users submitting jobs to the MDCS via the scheduler.
  - Use of interactive access on cheetah compute nodes could overload a node since this would effectively bypass the Cheetah cluster's grid engine scheduler.

- **Non-interactive Use:** Job&Task based
  - 2 main types: Independent vs Communicating Jobs
    - Both types can be used with either the local (on a non-cluster workstation) or MDCS profile.
Key usage scenarios: Algorithms

* Distributed/Parallel algorithm characteristics overview:
  
  – Small memory problems with no communication between workers
  
  – Small memory problems with some communication between workers
  
  – Large memory problems with no communication between workers
  
  – Large memory problems with some communication between workers

  – Variations on these fundamental characteristics
  
    • Which case does your code fit best?
Key usage scenarios: Process

- General Matlab Code Parallelization Process:
  - Develop serial Matlab code locally
  - Develop parallel Matlab code locally with Parallel Compute Toolbox using local configuration. When developing locally, Parallel Compute Toolbox can be used interactively: matlabpool, spmd, co-distributed arrays, pmode as part of algorithm development.
  - Code structure must be converted to non-interactive form in order to use the MDCS. You can develop your non-interactive code locally also.
    - Once you have your code in the MDCS deployable non-interactive functional form, you can submit to MDCS
      - Submit() function also works with local workers
  - Scale code with MDCS using the scheduler interface: submit(job)
    - *The scheduler(via submit() function) should be the only interface used to allocate MDCS workers.*
Key usage scenarios: Workloads

- **2 main types of workloads can be implemented with the MDCS:**
  - A job is logically decomposed into a set of tasks. The job may have 1 or more tasks, and each task may or may not have additional parallelism within it.

**CASE 1: Independent**

- Within a job the parallelism is fully independent, we have the opportunity to use MDCS workers to offload some of the independent work units. The code will not make use parallel language features such as parfor, spmd. Note: In many cases, parfor can be transformed into a set of tasks.
  - createJob() + createTask(), createTask(), … createTask()

**CASE 2: Communicating**

- Within a single job the parallelism is more complex, requiring the workers to communicate or when parfor, spmd, codistributed arrays (language features are used from Parallel Compute Toolbox).
  - createCommunicatingJob(), createTask()

  - Note: interactive use of the matlabpool/spmd command is not recommended on the MDCS since it will lock up workers and bypasses Matlab scheduler. However, using it with the local configuration on a workstation is a useful way to develop and test your algorithm.

  - Only 1 task can be created within a communicating job.

  - matlabpool command should never found in the source code submitted to MDCS.
MDCS Utilization Examples

- Custom parallel signal / image processing
  - Distributing a large set of 1-D signals or images pieces across multiple “labs” partitioning the domain.
  - When data is too large to fit on a single system, breaking up the data using distributed arrays allows memory scaling.
  - Processing requires workers to communicate with each other.

- SimBiology batch parameter processing
  - A toolbox that does not compile using Compiler Toolbox.
  - Allows processing many simulations as independent tasks without using up either regular Matlab licenses or SimBiology toolbox licenses.

- Optimization, Image Processing Toolbox, other toolboxes
  - Some functions/toolboxes have built in support for PCT + MDCS
  - Functions such as fft,max,min,prod,arrayfun,lu, svd,... work on co-distributed arrays(part of spmd).
Hands-on Training: Access

- You must first log in to the cheetah cluster, from here you can access the MDCS.
- From a Linux Desktop System: (e.g. the CBI Linux workstations)
- Access the shell, then
- Log into to `ssh -Y username@cheetah.cbi.utsa.edu`
- `qlogin`
- `matlab` &
Hands-on Training: Access

- From a Windows System: Access can be obtained using PuTTY + Xming
Hands-on Training: Access

- Within Putty, login to the CBI Cheetah Cluster with your CBI username password.
- **Immediately use qlogin to move from the head node to an interactive access node.**
- qlogin
- matlab &
The Matlab Interactive environment should come up on the compute node to which your interactive session was allocated.

This is the “client” session used as the interface to the MDCS.

In the current MDCS setup, you must access the MDCS exclusively via the Cheetah Cluster login process, for job submission purposes only.

Use the findResource function to obtain information about the available schedulers.

At the matlab command prompt type:

- `findResource('scheduler', 'type', 'jobmanager')`

   This will return key information needed to setup your environment:

   - Name: manager1
   - Hostname: compute-5-3.local
   - HostAddress: 192.168.255.201
   - ClusterSize: 64

This is a demo job manager name/info for the workshop.
Hands-on Training: Access

- Profile Setup: Using the information obtained with the findResource function, create a cluster profile for the job manager you were allocated.
Hands-on Training: Access

- Create a new MJS profile:
Hands-on Training: Access

- Update the following key fields:

  - **Description of this cluster (Description):**
    - Hostname of the machine where MJS is running. (Normally needed)

  - **Host:**
    - compute-5-3.local

  - **MJSName:**
    - manager1

  - **Username:**
    - current user (default)

  - **Files and Folders:**
    - Files and folders to copy from client to cluster nodes (One entry per line)
      - AttachedFiles

      `<name>`
Hands-on Training: Access

- Username/Password Setup: The system will ask you to create a username and password for the MDCS. The default should be your CBI username. Note: The MDCS username/password is not tied to your cbi username/password.
Hands-on Training: Access

- Click on Validate to make sure you have full functionality:
Hands-on Training: Access

Validation details:

- **Stage: Cluster connection test (parcluster)**
  - Status: Passed
  - Description: Validation Passed
  - Command Line Output: (none)
  - Error Report: (none)
  - Debug Log: (none)

- **Stage: Job test (createJob)**
  - Status: Passed
  - Description: Validation Passed
  - Command Line Output: (none)
  - Error Report: (none)
  - Debug Log: (none)

- **Stage: SPMD job test (createCommunicatingJob)**
  - Status: Passed
  - Description: Validation Passed
  - Command Line Output: (none)
  - Error Report: (none)
  - Debug Log: (none)

- **Stage: Pool job test (createCommunicatingJob)**
  - Status: Passed
  - Description: Validation Passed
  - Command Line Output: (none)
  - Error Report: (none)
  - Debug Log: (none)

- **Stage: MATLAB pool test (matlabpool)**
  - Status: Passed
  - Description: Validation Passed
  - Command Line Output: (none)
  - Error Report: (none)
  - Debug Log: (none)

- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.
- Received a socket connection.

Did not find any pre-existing parallel jobs created by matlabpool.

Error Report: (none)
Debug Log: (none)
Hands-on Training: Access

- System Ready: You are now ready to submit jobs to the CBI MDCS.
Hands-on Training: Access

- First example: Having the MDCS generate a random matrix.

```matlab
g% = parcluster('Profile1');
job = createJob(g%);
createTask(job, @rand, 1, {100, 100});
send(job);
wait(job);
resultCellArray = job.fetchOutputs;
resultImage = resultCellArray{1}
imagesc(resultImage)
```
Hands-on Training: Access

- Job Cleanup: Make sure to cleanup the completed jobs on the MDCS.
- Use the CBI wrapper script to clean after you finish both local mode or MDCS mode: `cbi_deletemdcsjobs('test.user','jobmanager','manager1','completed')`
Hands-on Training: Access

- Job Cleanup: Make sure to cleanup the completed jobs on the MDCS.
- Use the CBI wrapper script to clean after you finish both local mode or MDCS mode.
Hands-on Training: Development Modes

Local Profile
Thinkpad W520
(Intel i7 Processor)
Windows 7
Hands-on Training: Development Modes

MATLAB R2012a

Job Monitor

Select Profile: Profile1

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Submit Time</th>
<th>Finish Time</th>
<th>Number of tasks</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:13:21 CDT 2</td>
<td>Thu Sep 06 10:13:36 CDT 2</td>
<td>16</td>
<td>finished</td>
</tr>
<tr>
<td>141</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:18:08 CDT 2</td>
<td>Thu Sep 06 10:18:22 CDT 2</td>
<td>20</td>
<td>finished</td>
</tr>
<tr>
<td>142</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:18:40 CDT 2</td>
<td>Thu Sep 06 10:18:53 CDT 2</td>
<td>20</td>
<td>finished</td>
</tr>
<tr>
<td>143</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:52:23 CDT 2</td>
<td>Thu Sep 06 10:52:37 CDT 2</td>
<td>20</td>
<td>finished</td>
</tr>
<tr>
<td>144</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:34:10 CDT 2</td>
<td>Thu Sep 06 10:34:22 CDT 2</td>
<td>24</td>
<td>finished</td>
</tr>
<tr>
<td>145</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:34:33 CDT 2</td>
<td>Thu Sep 06 10:34:45 CDT 2</td>
<td>24</td>
<td>finished</td>
</tr>
<tr>
<td>146</td>
<td>nelson.ramirez</td>
<td>Thu Sep 06 10:35:20 CDT 2</td>
<td>Thu Sep 06 10:35:32 CDT 2</td>
<td>24</td>
<td>finished</td>
</tr>
</tbody>
</table>

Last updated at Thu Sep 06 10:37:28 CDT 2012

Command Window

Elapsed time is 15.82736 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 15.601382 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 14.87190 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 14.34039 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 14.474045 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 13.030751 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 13.305140 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 12.819768 seconds.
>> mdcsworkshopdemo(7)
Elapsed time is 12.819977 seconds.
>>

MDCS Profile
Hands-on Training: Example

- Moving code from script form to non-interactive functional form ready to submit to the MDCS using the job interface.
  - License benefits
  - Performance analysis:
    - Local vs. MDCS
    - Parallelism Tradeoffs
- Part 1: Serial Mandelbrot Image Generator
- Part 2: Interactive mode parfor Mandelbrot Image Generator
- Part 3: Interactive mode parfor (slicing issue) Mandelbrot Image Generator
- Part 4: Moving to job/task based workflow (intro case)
- Part 5: Moving Mandelbrot script to functional form + job/task workflow
- Part 6: Many communicating jobs (parameter sweeps)
- Part 7: Many independent tasks (1 job) (parameter sweeps)
Hands-on Training: Example

Mandelbrot set:

Fundamental image generation algorithm has multiple levels of parallelism.

From the parallelism at the pixel level, to the high level parallelism when generating many images with different parameters.
Hands-on Training: Example

Mandelbrot(parfor) Parallelism (1 image with 1 communicating job + 1 task)
Local(Thinkpad W520 Intel i7 @2.40 GHz) vs.
MDCS(Dell PowerEdge M910 Intel Xeon x7560 @ 2.26 GHz)
Hands-on Training: Example

Parallel(Communicating) Job Scaling: Examples 5&6

Local vs. MDCS + 1 vs 5 jobs (one mandelbrot image per job)

- Example5, 1 job, Local- i7 Thinkpad W520
- Example5, 1 job, MDCS
- Example6, 5 jobs, Local- i7 Thinkpad W520
- Example6, 5 jobs, MDCS

Graph showing the rate (images/second) vs. the number of labs/workers per job.
Hands-on Training: Example

Mandelbrot: 1 job + Independent tasks: Example 7

Local vs. MDCS Total Runtime

- Local: i7 Thinkpad W520
- MDCS

Graph showing runtime (seconds) vs. number of images (tasks) for Local and MDCS.
Mandelbrot: 1 job + Independent tasks: Example 7

Local vs. MDCS
Rate (Images/Sec)

Local- i7 Thinkpad W520
MDCS
Hands-on Training: Example
Hands-on Training: Example
Summary

- Initial R2012a MDCS setup available @ CBI
- Great opportunity to use a cutting edge parallel software development infrastructure
  - R2012a has many improvements over R2011 version
- The World of Parallel Programming is available within Matlab
  - Both Distributed and Parallel workloads
  - Benefit from license usage minimization, scaling distributed & parallel codes(compute+memory), high-throughput workflow enablement
- Planned improvements to our setup via Sun Grid Engine Integration
  - Authentication improvements planned:
    - Currently authentication to MDCS is not tied to CBI authentication, except via job submission process. Users must submit work to MDCS from within a cheetah cluster compute node.
    - Shared file system access vs. Transferring data to and from MDCS nodes
  - Moving to an Infiniband back-end network for improved performance
- Use case expansion: Many opportunities to use the MDCS across many fields.
Key Points

**MDCS Benefits:**

1) Matlab License Usage Minimization

2) Performance Scaling (Compute+Memory)

- Interested in using the PCT+MDCS for your project? Schedule a consulting meeting with us.

- We look forward to hearing from you!
  - [http://cbi.utsa.edu/staff](http://cbi.utsa.edu/staff)
Appendix
Future Work

**MDCS Part2: Applied usage examples**
- SimBiology + task based parallelism w/MDCS
- Optimization Toolbox parallelism examples
- Image processing Toolbox parallelism: Batch template matching, block level parallelism
- Custom Communicating jobs
- Deployed Matlab executables with MDCS access

**Matlab+PCT+MDCS+Compiler Toolbox Items of Interest**
- Memory scaling with MDCS
- Moving past contiguous memory allocation limits
- Single vs Double precision issues
- Implicit vs. Explicit Matlab multithreading, Deployed Matlab executable explicit parallelism with PCT, .. many more items of interest
Future Work

- **Matlab + External Library integration via MEX interface (C/C++/Fortran w/OpenMP)**
  - We developed an approach for optimized performance + additional functionality:
    - Performance improvement via low level optimization + parallelism with C++/OpenMP
  - Handling single vs double precision within MEX interface
  - GSL, ITK, OpenCV, CUDA external library integration
  - Linux, Windows, Mac Deployment examples: Creating new high performance multi-threaded, multi-platform Matlab toolboxes in C++

- **Intro to General Purpose Graphical Processing Units & Matlab integration**
  - Using Parallel Compute Toolbox GPU features
  - Integrating custom CUDA kernels within Matlab
  - Integrating C++ code that uses GPU's directly with Thrust + Cusp C++ Libraries
CBI Laboratory

http://cbi.utsa.edu

Development team:

Zhiwei Wang, Director

David Noriega, Jean-Michel Lehker, Nelson Ramirez
Acknowledgments

- Dr. Toma Marinov (Post-Doc Researcher) with Dr. Fidel Santamaria's Laboratory, whose utilization of the MDCS@CBI has provided valuable feedback for this workshop.

- Mr. Yung Lai for his help testing client application codes with the MDCS CBI infrastructure.
References


[9] http://www.psc.edu/index.php/matlab (MDCS @ PCS)


References

[19] http://www.psc.edu/index.php/matlab (MDCS @ PSC)

[20] http://rcc.its.psu.edu/resources/software/dmatlab/ (MDCS @ Penn State)

[21] http://ecr.buffalo.edu/support/software-resources/compilers-programming-languages/matlab/mdcs.html (MDCS @ U of Buffalo)


[28] http://www.hpc.maths.unsw.edu.au/tensor/matlab (MDCS @ UNSW)


[30] http://www.rcac.purdue.edu/userinfo/resources/peregrine1/userguide.cfm#run_pbs_examples_app_matlab_licenses_strategies (MDCS @ Purdue)


[32] http://www.nccs.nasa.gov/matlab_instructions.html (MDCS @ Nasa)

[33] http://www.mathworks.com/help/toolbox/distcomp/m/bs8h9g9-1.html (PCT, MDCS R2012a interface changes)


References


[38] http://www.hpc.fsu.edu/ (MDCS @ FSU, HPC main site)

