Distributed Computing User's Guide

Johannes Gutenberg-Universität Mainz
Alexey Cherepnev[*]

April 26, 2011

1 Overview

Parallel Computing Toolbox and MATLAB Distributed Computing Server software let you solve computationally and data-intensive problems using MATLAB and Simulink on multicore and multiprocessor computers. Parallel processing constructs such as parallel for-loops and code blocks, distributed arrays, parallel numerical algorithms, and message-passing functions let you implement task-parallel and data-parallel algorithms at a high level in MATLAB without programming for specific hardware and network architectures.
Some Definitions:
A **job** is some large operation that you need to perform in your MATLAB session. A job is broken down into segments called **tasks**. You decide how best to divide your job into tasks. You could divide your job into identical tasks, but tasks do not have to be identical.

The MATLAB session in which the job and its tasks are defined is called the **client** session. Often, this is on the machine where you program MATLAB. The client uses Parallel Computing Toolbox software to perform the definition of jobs and tasks. The MATLAB Distributed Computing Server product performs the execution of your job by evaluating each of its tasks and returning the result to your client session.

The **job manager** is the part of the server software that coordinates the execution of jobs and the evaluation of their tasks. The job manager distributes the tasks for evaluation to the server's individual MATLAB sessions called **workers**. Use of the MathWorks job manager is optional; the distribution of tasks to workers can also be performed by a third-party scheduler, such as Window HPC Server (including CCS), a Platform LSF scheduler, or a PBS Pro scheduler.

2 Job Managers, Workers, and Clients

The optional job manager can run on any machine on the network. The job manager runs jobs in the order in which they are submitted, unless any jobs in its queue are promoted, demoted, canceled, or destroyed.

Each worker receives a task of the running job from the job manager, executes the task, returns the result to the job manager, and then receives another task. When all tasks for a running job have been assigned to workers, the job manager starts running the next job with the next available worker.

3 mdce Service

If you are using the MathWorks job manager, every machine that hosts a worker or job manager session must also run the mdce service.

[*]cherepne@uni-mainz.de

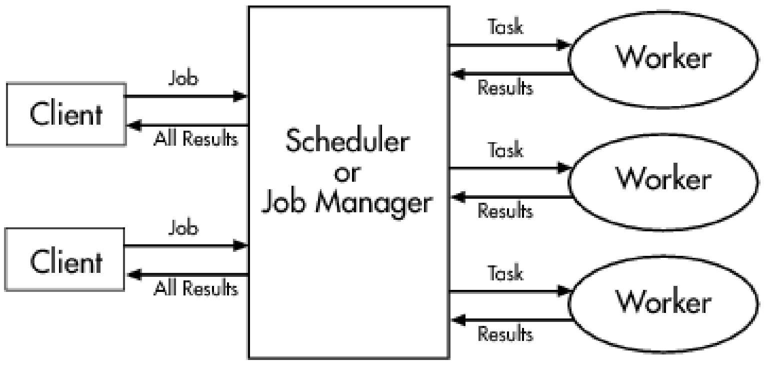

Figure 1: Client—Job-Manager—Worker Architecture

The mdce service recovers worker and job manager sessions when their host machines crash. If a worker or job manager machine crashes, when mdce starts up again (usually configured to start at machine boot time), it automatically restarts the job manager and worker sessions to resume their sessions from before the system crash.

4 Using Parallel Computing Toolbox Software

A typical Parallel Computing Toolbox client session includes the following steps:

1. Find a Job Manager (or scheduler) — Your network may have one or more job managers available (but usually only one scheduler). The function you use to find a job manager or scheduler creates an object in your current MATLAB session to represent the job manager or scheduler that will run your job.

2. Create a Job — You create a job to hold a collection of tasks. The job exists on the job manager (or scheduler's data location), but a job object in the local MATLAB session represents that job.

3. Create Tasks — You create tasks to add to the job. Each task of a job can be represented by a task object in your local MATLAB session.

4. Create Tasks — You create tasks to add to the job. Each task of a job can be represented by a task object in your local MATLAB session.

5. Submit a Job to the Job Queue for Execution — When your job has all its tasks defined, you submit it to the queue in the jobmanager or scheduler. The job manager or scheduler distributes your job's tasks to the worker sessions for evaluation. When all of the workers are completed with the job's tasks, the job moves to the finished state.

6. Retrieve the Job's Results — The resulting data from the evaluation of the job is available as a property value of each task object.

7. Destroy the Job — When the job is complete and all its results are gathered, you can destroy the job to free memory resources.

5 Planning Your Network Layout

Worker sessions usually run on the cluster of machines dedicated to that purpose. The MATLAB client session usually runs where MATLAB programs are run, often on a user's desktop.

The job manager process should run on a stable machine, with adequate resources to manage the number of tasks.

The following table shows what products and processes are needed for each of these roles in the parallel computing configuration: Session— Product— Processes Client— Parallel Computing Toolbox —MATLAB with toolbox —Worker —MATLAB Distributed Computing Server—worker; mdce service Job manager —MATLAB Distributed— Computing Server—mdce service, job manager

Note that the parallel computing products do not provide any security measures.

6 Stopping the Job Manager and Workers. Microsoft Windows Operating Systems

Enter the commands of this section at the prompt in a DOS command window. To shut down the job manager, enter the commands
cd matlabroot\toolbox\distcomp\bin
Enter the following command on a single line:
stopjobmanager -remotehost "job manager hostname" -name "MyJobManager" -v
If you have more than one job manager running, stop each of them individually by host and name. For a list of all options to the script, type
stopjobmanager -help

For each MATLAB worker you want to shut down, enter the commands
cd matlabroot\toolbox\distcomp\bin
stopworker -remotehost "worker hostname" -name "worker name" -v If you have more than one worker session running, you can stop each of them individually by host and name.
stopworker -remotehost "worker hostname" -name "worker1 name"
stopworker -remotehost "worker hostname" -name "worker2 name"
For a list of all options to the script, type
stopworker -help

7 Starting in a Clean State

When a job manager or worker starts up, it normally resumes its session from the past. This way, a job queue is not destroyed or lost if the job manager machine crashes or if the job manager is inadvertently shut down. To start up a job manager or worker from a clean state, with all history deleted, use the "-clean" flag on the start command:
startjobmanager -clean -name MyJobManager
startworker -clean -jobmanager MyJobManager

8 The Simplest Example

Suppose we have code which calculate the result of a function locally using 8 workers (we do not need Distributed Computing Toolbox for it) and *par-for* command. One need to do the following steps:

1. Check whether *mdce* service is working.

2. Run
   cd matlabroot\toolbox\distcomp\bin admincenter.bat.

3. Start a new jobmanager (called "*jobmanager*" later to refer to the name) and start n workers ($n = 1, 2, \ldots 16$ in our architecture) associated to the *jobmanager*.

4. Start Matlab and create a new configuration ("Parallel"$\rightarrow$ "Manage Configurations"). To do so choose "File"$\rightarrow$"New"$\rightarrow$"Job Manager". Fill the name "*jobmanagerconfiguration*", job manager host name "*sophia.vwl.uni-mainz.de*", job manager name "*jobmanager*".

5. Go to list "*Jobs*" and add directories which should be used to compute the result, e.g. containing functions. Here one can also limit the number of cores he is going to use. Logically it should be less than the number of workers associated with the job manager.

6. Validate "*jobmanagerconfiguration*".

7. The main code is

   ```
   clear
   clc
   truevalue = 1:32;
   matlabpool('open', 'jobmanagerconfiguration', 16)
   tic
   parfor schleife = 1:32
    estimate(schleife) = simu(truevalue(schleife));
   end
   toc
   matlabpool('close')
   ```

8. Function "*simu*" should be written in the following way:

   ```
   function out = simu(truevalue)
   for i=1:4*10^9
       1+1;
   end
       out = truevalue^2;
   end
   ```

Results

for 16 workers 100% of processor usage and:

```
Starting matlabpool using the 'jobmanagerconfiguration' configuration ...
connected to 16 labs.
Elapsed time is 52.154761 seconds.
Sending a stop signal to all the labs ... stopped.
```

for 8 workers 50% of processor usage and:

```
Starting matlabpool using the 'jobmanagerconfiguration' configuration ...
connected to 8 labs.
Elapsed time is 103.680432 seconds.
Sending a stop signal to all the labs ... stopped.
```