High-Throughput Computing Use Cases

August 16, 2019

Version 2.1

These use cases describe the most common ways that researchers use high-throughput computing (HTC) resources. Also known as "capacity computing," high-throughput computing allows researchers to run a very large number of modeling, simulation, or data analysis tasks in a short amount of time. The resources required by each task can be small (similar to what one could run on a basic computer) or very large (requiring more active memory or CPU cycles than "normal" computers provide). What makes HTC unique is the fact that many runs of the same application are required to complete a research project. HTC resources allow researchers to complete projects faster than they could if they were limited to the computers available in their own offices, labs, or research institutions.

Each HTC resource is designed, constructed, and operated by a *service provider (SP)* organization, such as the Pittsburgh Supercomputing Center (PSC) or University of Michigan. An HTC resource may contribute services to one or more *public research computing communities*, such as XSEDE or Open Science Grid (OSG). These use cases focus on the experiences researchers in a given community have with the HTC resources in that community. Using an HTC resource is most often part of a larger research process, so these use cases also mention the community's website and documentation, registration and account management services, allocation services, and tools to help move applications and data into and out of individual HPC resources.

HTC-01: Run a set of independent jobs on an HTC resource

A **researcher** needs to run an application many times with varying inputs on a single community high-throughput computing (HTC) resource. Each application run is called a "job." The complete set of jobs is called a "project." Once jobs are specified and submitted to the resource, the resource executes them without further involvement by the researcher.

We assume the following things.

1. The researcher is registered with the community and manages a project that has received an allocation on the HTC resource.
2. The project allocation is based on an existing application and reasonable estimates for the required computing resources (CPU hours, RAM, storage, etc.).
3. The application is already installed and validated on the HTC resource.
4. The inputs to every job are available before any jobs are submitted. Jobs can be run in any order or simultaneously.

In most cases, the researcher expects it to work as follows.

1. First, the researcher studies the documentation for the HTC resource to understand the resource's basic interface.
2. Then, the researcher logs into a login node for the HTC resource.
3. Then, the researcher uses a script or program to generate a set of job definitions.
4. Then, the researcher invokes one or more job submissions commands to submit the generated job definitions to the HTC resource.
5. The HTC resource validates, queues, and executes the jobs. Each job reads its inputs and produces outputs.
6. The researcher monitors the status of jobs and their outputs.
7. The researcher may cancel the job set, causing any remaining jobs to be removed from the queue.

It will always be like this, except when the following are true instead.

1. A **software developer** needs to develop an application or science gateway that can specify and submit jobs to the high-throughput computing (HTC) resource. In this case, Steps 2-4 are replaced by developing the application. The application uses the remote job submission feature described in use case CAN-01. Steps 6-7 are also handled by the application rather than a human being.
2. The "application" is actually a script that runs multiple commands. Each command might be an application run. In this case, the job definitions described in Steps 3-4 reference the script, and when each job runs, the script is executed, which may result in multiple application runs. The script is responsible for managing the inputs and outputs of each application run.

We'll take any solution, as long as the following are true.

1. The researcher can use the solution to process at least a million jobs per project.
2. Status information for each job is available. The status information includes: error messages, exit codes, stdout/stderr.

3. The researcher can determine the overall status of the project's jobs in under 60 seconds.
4. The researcher can perform all needed job tracking functions on a login host of the HTC resource.

HTC-02: Run a set of interrelated jobs on an HTC resource

A **researcher** needs to run one or more applications many times with varying inputs on a single community high-throughput computing (HTC) resource. Each application run is called a "job." The complete set of jobs is called a "project." Some jobs may use the outputs of other jobs as their inputs. (This is called a "dependency.") Once the jobs and dependencies are specified to the system, the system executes them without further involvement by the researcher.

We assume the following things.

1. The researcher is registered with the community and manages a project that has received an allocation on the HTC resource.
2. The project allocation is based on existing applications and reasonable estimates for the required computing resources (CPU hours, RAM, storage, etc.).
3. The applications are already installed and validated on the HTC resource.

In most cases, the researcher expects it to work as follows.

1. The researcher generates a workflow description that defines the project's jobs and the dependencies between jobs.
2. The researcher submits the workflow description to the system.
3. The system submits jobs to the HTC resource until the project is complete.
4. While jobs are being processed by the HTC resource, the system tracks the status and completion of jobs, releasing dependent jobs and performing job throttling and retry as needed.
5. While jobs are being processed by the HTC resource, the researcher monitors the status of jobs and their outputs.
6. While jobs are being processed by the HTC resource, the researcher may cancel the job set, causing any remaining jobs to be removed from the system.

It will always be like this, except when the following are true instead.

1. A **software developer** needs to develop an application or science gateway that can specify and submit workflow descriptions to the system. In this case, Steps 1-2 are replaced by developing the application. The application uses an API provided by the system for submitting the workflow description and any job inputs and for collecting job outputs. Steps 5 and 6 are also handled by the application rather than a human being.

We'll take any solution, as long as the following are true.

1. The researcher can use the solution to process several million jobs per project.
2. Status information for each job is available. The status information includes: error messages, exit codes, stdout/stderr.
3. The researcher can determine the overall status of the project's jobs in under 60 seconds.

4. The researcher can perform all needed job tracking functions on a login host of the HTC resource.

HTC-04: Run a set of interrelated jobs on several HTC resources

A **researcher** needs to run one or more applications many times with varying inputs on several community high-throughput computing (HTC) resources. Each application run is called a "job." The complete set of jobs is called a "project." Some jobs may use the outputs of other jobs as their inputs. Some jobs may need to run on the same resource as other jobs, possibly at the same time. (These are called "dependencies.") Once the jobs and dependencies are specified to the system, the system executes them without further involvement by the researcher.

We assume the following things.

1. The researcher is registered with the community and manages a project that has received allocations on each of the HTC resources.
2. The project allocations are based on existing applications and reasonable estimates for the required computing resources (CPU hours, RAM, storage, etc.).
3. The applications are already installed and validated on each of the HTC resources.

In most cases, the researcher expects it to work as follows.

1. The researcher generates a workflow description that defines the project's jobs and the dependencies between jobs.
2. The researcher submits the workflow description to the system.
3. The system submits jobs to the HTC resources until the project is complete.
4. While jobs are being processed by the HTC resource, the system tracks the status and completion of jobs, releasing dependent jobs and performing job throttling and retry as needed.
5. While jobs are being processed by the HTC resource, the researcher monitors the status of jobs and their outputs.
6. While jobs are being processed by the HTC resource, the researcher may cancel the job set, causing any remaining jobs to be removed from the system.

It will always be like this, except when the following are true instead.

1. A **software developer** needs to develop an application or science gateway that can specify and submit workflow descriptions to the system. In this case, Steps 1-2 are replaced by developing the application. The application uses an API provided by the system for submitting the workflow description and any job inputs and for collecting job outputs. Steps 5 and 6 are also handled by the application rather than a human being.

We'll take any solution, as long as the following are true.

1. The researcher can use the solution to process several million jobs per project.
2. Status information for each job is available. The status information includes: error messages, exit codes, stdout/stderr.
3. The researcher can determine the overall status of the project's jobs in under 60 seconds.

4. The researcher can perform all needed job tracking functions on a login host of one of the HTC resources.
5. The system can work with up to sixty HTC resources for a single project.

History

	Version	Date	Changes	Author
Entire Document	0.1	10/25/2012	First Version submitted to A&D	M. Wilde
	0.2	2/26/2013	Revised based on discussion with Altaf.	M. Wilde
	0.3	5/31/2013	First draft of use cases 1.1 through 2.1	M. Wilde
	0.4	8/15/2013	Various technical edits	Foster, Grimshaw, Tuecke, Wilde
	2.0	3/13/2014	Final XSEDE-1 version; minor formatting and typographical edits	Foster, Grimshaw, Hossain, Wilde
Entire document	2.1	8/16/2019	Merged HTC-02 and HTC-03 because their only difference was implementation, not user experience; reformatted to XSEDE-2 format; standardized terminology with other use case areas; removed unnecessary XSEDE terminology	L. Liming