Validating and Certifying Clusters with Intel® Cluster Checker

César Martínez Scott McMillan Ricardo Medel

Intel Corp.
{cesar.martinez,scott.a.mcmillan,ricardo.medel}@intel.com

Abstract. New users of cluster computing find a high barrier to entry due the daunting complexity of deploying and manage a cluster. Intel® Cluster Ready program aims to lower the entry barrier and set standards for the cluster integration process. This paper describes the Intel® Cluster Checker tool, a key component of the Intel Cluster Ready program. The Cluster Checker is an automated and flexible script that validates the cluster settings against the Intel Cluster Ready specification and checks the general wellness of the cluster.

Keywords: clusters, high-performance computing, Perl, validation

1. INTRODUCTION

In recent years, cluster computing has emerged as a scientific means for obtaining additional computational power in the commercial (manufacturing and services), health, finance, and educational areas [1, 2, 5]. The comparatively low cost and scalability of clusters is pushing more companies to enter in the cluster's arena. However, the capability of each cluster involves an exchange of hardware costs for software costs [3]. A complex combination of software is required for configure and maintain the distributed heterogeneous machines that make up the cluster. Therefore, the new users still have a high barrier to entry since they are forced to do extensive research in order to establish hardware and software requirements, build the cluster, install the required software, and tune the components to obtain the desired performance.

Given the complexity of the task of building a cluster, different users developed different *ad-hoc* procedures for cluster deployment and management, but the associated lack of standards keeps the cluster ownership costs high. To ameliorate this situation, some of the "best practices" techniques for cluster deployment were integrated in easy-to-use toolkits called *provisioning systems* [3, 4]. However, the solution offered by provisioning systems only considers certain specific software stack over a generic hardware. As a result of this approach, the cluster integrator should find by himself the way of obtain the best potential performance from his hardware.

In order to overcome the limitations of the provisioning systems, Intel developed the Intel® Cluster Ready (ICR) program to simplify the design, building, and deployment of clusters based on Intel components. A key component of this program is the Intel® Cluster Checker tool. The Cluster Checker is a powerful and flexible tool for helping customers to integrate clusters.

2. INTEL CLUSTER READY PROGRAM

The ICR program is a collaborative effort between Intel, Original Equipment Manufacturers (OEMs), channel members, and Independent Software Vendors (ISVs) in order to establish a specification to be used as a common basis for performing clusters. Clusters that comply with this specification are certified by Certifying Entities of the ICR program. Also, applications produced by ISVs can be certified as complying with the ICR specification. The main idea behind this certification process it that applications written to run on one certified cluster can reliably run on any certified cluster. Conversely, a certified cluster will be able of running any certified application.

In order to certify a cluster, the system integrator must first deploy it following the steps indicated by a Cluster Reference Implementation or recipe. These recipes are the product of a cluster engineering process that establishes the steps required for integrate a cluster. The next step is to use the Cluster Checker to verify the cluster settings. Finally, the manual completion of a check list ensuring that certain procedures were followed during the deployment phase is required for the certification process.

3. INTEL CLUSTER CHECKER

The Cluster Checker is a software tool that helps to verify the conformance of clusters to the ICR specification. It can verify the cluster at node and cluster-wide levels, ensuring the cluster's nodes are uniformly and optimally configured. This tool is customizable and extensible, letting users to customize tests by specifying commands to be executed and the expected output, and even integrating additional test.

The tool is written in Perl. In a nutshell, it consists of an engine that loops over a set of test modules or *checks* (see Figure 1.) Prior to entering the loop, depending on the cluster configuration and the verification mode, the engine must figure out which test modules to run, which nodes to check, and setup the infrastructure to process the test modules.

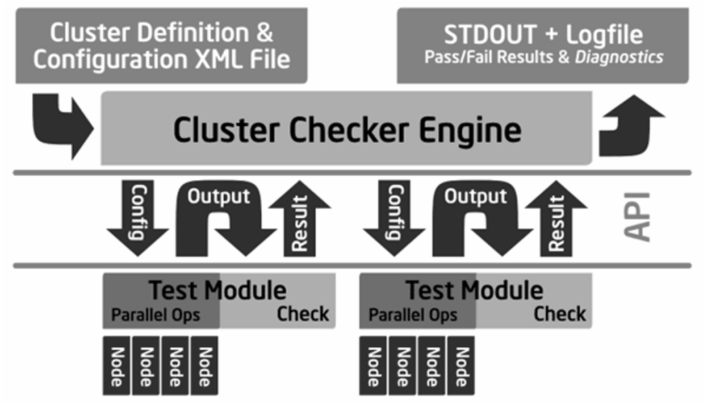

Figure 1. Test Modules Execution flow.

Cluster Checker includes 106 checks. Checks are Perl modules that do not execute on the cluster nodes: each module runs one or more commands on a node. So, it is not needed to have a Perl interpreter at each node of the cluster. A module has two functional steps: first collect information and then determine if the information is 'correct.' 'Correct' may mean matching a certain value, uniformity compared to other values, or something else. Thus, modules must have a 'gather' step where it collects information and a 'test' step where it determines if that information is 'correct.'

There are four different classes of modules to handle special cases of this two-step gather/test process, but every class maintains this division. The classes of modules are *unit* (checks correctness of a node property compared to a value), *vector* (checks the uniformity of a node property across the cluster), *span* (checks a cluster-wide property compared to a value), and *matrix* (checks a node-to-node or pair-wise property for every possible node-to-node combination).

There are two modes of checking: the *wellness* checking of a cluster and the *compliance* mode with respect to the ICR Specification. Wellness checking consist a set of test verifies the functional and non-functional characteristics of a cluster like the performance or disk space. It can be used in order to test on demand the cluster. This feature of the Cluster Checker helps to maintain a cluster by providing diagnostic data to identify potential issues. Compliance checking, on the other hand, is used during the certification phase. The goal of applying this checking is to determine if the cluster configuration follows the ICR Specification.

4. CONFIGURING INTEL CLUSTER CHECKER

The Cluster Checker allows the configuration of three aspects of the verification procedure: the architecture of the cluster, the configuration of individual test modules, and the runtime behavior of the tool.

4.1. Cluster Architecture Configuration

Clusters are typically composed by many individual nodes. Some nodes control the cluster, others are used as computational resources, and others may be used for different purposes, such as storage servers. Therefore, some checks make sense only on nodes of a certain type while other may behave differently depending on the node type. The Cluster Checker recognizes three functional types of nodes: *compute*, *head*, and *other*. A text configuration file is used to establish the name of each node and its type.

Heterogeneous clusters are recognized by Cluster Checker using the *group* concept. A group defines a subset of nodes with common characteristics, such as the same amount of memory or the same processor. The text configuration file can be used to assign nodes to arbitrary groups.

4.2. Test Module Configuration

The default settings of the test modules are appropriate in most cases, but may not be appropriate for all clusters. Consequently, the most valuable check will be one that is optimized for the cluster being validated.

Individual checks may be configured by specifying values in a configuration file written in XML. Moreover, different values can be specified for each node group.

4.3. Tool Behavior Configuration

The Cluster Checker allows modifying the set of modules to execute by explicitly excluding or including checks. The default list of checks may be altered at runtime by including the <exclude_module>, <include_module>, and <include_only_module> tags in the XML configuration file.

Checks form a hierarchy that build up from simple to complex. This is accomplished through dependencies by one module on others; if a check fails, all checks that are dependent on it will be skipped. The module dependencies may be modified by adding or removing dependencies with the <add_dependency> and <remove_dependency> tags.

The *matrix* checks can produce a combinatorial explosion that can be controlled by using the <alltoall-throttle> tag. This tag allows specifying a value indicating the degree of neighbors to be used to generate pairs, e.g. a value of 2 checks each node with its nearest and next-nearest neighbors. Node neighbors are determined by position in the nodelist file, not the physical arrangement of the nodes.

Checks are parallelized by forking a process for each node. The default number of nodes that can be simultaneously checked (64) can be override by using the <process-limit> tag.

5. CONCLUSIONS

The growth of the cluster market is demanding standards for cluster integration that simplify the tasks of designing, building, and tuning a cluster. The Intel® Cluster Ready (ICR) program is a solution to this need; its goal is to establish a specification to be used as a common basis for performing clusters.

The Intel® Cluster Checker tool provides automated software for verifying the configuration of clusters. It can be used as part of the ICR program to validate the cluster's compliance with the ICR specification, and it can be used independently to help the cluster integrator to obtain the best of the hardware and software stack. The tool is composed by 106 checks that verify the cluster at node and cluster-wide levels, looking for CPU, firmware, kernel, storage, and network settings known to slow down node performance and ensuring the cluster's nodes are uniformly and optimally configured. Moreover, the tool is extensible, letting users to customize tests by specifying command and expected output, and even integrating additional test via plug-ins.

More checks, automatic patching of the cluster, and a "maintenance" mode (issuing warnings if the last cluster's update crossed some performance thresholds) are features to be included in the next versions of this tool.

6. REFERENCES

[1] DE Editors, "*HPC Clusters Gain Momentum across all HPC Segments*", Desktop Engineering Online, December 2007.

[2] Earl C. Joseph, Steve Conway, Richard Walsh, Jie Wu, Daniel Lee, "*Worldwide Technical Computing Server 2008 Top 10 Predictions*", IDC Corp., January 2008.

[3] John Mugler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, Jeffrey M. Squyres, Benoît des Ligneris, Francis Giraldeau, Chokchai Leangsuksun, "*OSCAR Clusters*", Linux Symposium 2003, Ottawa, Canada, June 2003.

[4] Philip M. Papadopoulos, Mason J. Katz, Greg Bruno, "*NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters*", in "Concurrence and Computation: Practice and Experience", vol. 15, number 7-8, pages 707-725, 2003.

[5] Earl C. Joseph, "*HPC Cluster Market Trends*", IDC Corp., May 2007.