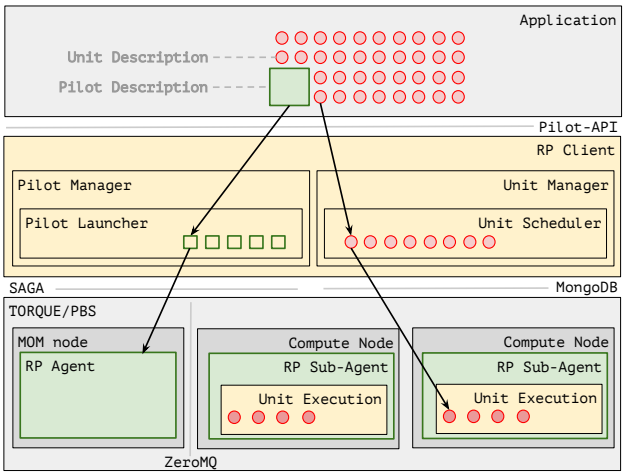
Executing dynamic heterogeneous workloads on Crays with RADICAL-Pilot

1 Introduction

Traditionally high-performance computing (HPC) systems such as Cray's have been designed primarily to support monolithic workloads - a single parallel application running across 100-1000s of compute nodes. However, the workload of many important scientific use-cases is instead composed of spatially and temporally heterogeneous tasks that are often dynamically inter-related. For example, simulating the dynamics of complex macromolecules is often done using "swarms" of short molecular dynamic calculations, each running on a small number of processors. The output of these calculations is collected to determine the next set of simulations. Such workloads still benefit from execution at scale on HPC resources but a tension exists between the workload's resource utilization requirements and the capabilities of the HPC system software and usage policies. Pilot systems have the potential to successfully relieve this tension.

2 RADICAL-Pilot

We describe RADICAL-Pilot (RP), a scalable and interoperable pilot system that implements the Pilot abstraction to support the execution of diverse workloads. We specifically describe the design and architecture (see figure) and characterize the performance of RP's task execution components, which are engineered for efficient resource utilization while maintaining the full generality of the Pilot abstraction. RP has been successfully used on Crays such as Blue Waters (NCSA), Titan (ORNL), Hopper & Edison (NERSC) and ARCHER (EPSRC), but also on IBM's Blue Gene/Q, many of XSEDE's HPC resources, and on the Open Science Grid (OSG).



RADICAL-Pilot Architecture. Pilots (description and instance) in green are for resource allocation and Units (description and instance) in red are for task execution. Applications interact with RP through the Pilot-API. Resource interoperability comes through SAGA. Client-Agent communication via MongoDB. Inter-Agent communication via ZeroMQ.

3 Enabling RP on Cray systems

To enable RP on Cray systems we have developed three ways of interfacing RP and the Cray system software.

3.1 Application Level Placement Scheduler (ALPS)

The ALPS system provides launch functionality for running executables on compute nodes in the system, interfaced with the "aprun" command. ALPS is the native way to run applications on a Cray from the batch scheduling system. By default, ALPS limits the user to run 1000 applications concurrently within one batch job, while in practice we also see values of only 100 concurrent applications. In the Pilot use-case, these applications may run only for a very short time, which puts further strain on ALPS and the MOM node and effectively limits the throughput of these applications. ALPS also doesn't allow the user to easily run more than one tasks on a single compute node, which makes it unattractive to launch workloads with heterogeneous application size.

3.2 Cluster Compatibility Mode (CCM)

Crays are effectively MPP machines and the Cray Compute Node OS does not provide a full set of the Linux services compared to typical Beowulf clusters. CCM is a software solution that provides those services when required by applications. Limitations of CCM include the unavailability of swap space. More importantly, it is not generally available on all Cray installations. Access to CCM varies per system, requiring special flags to the job description or submitting to a special queue (RP hides those differences from the application). RP can operate in CCM with the Agent either external or internal to the created CCM cluster. When the Agent is external it uses "ccmrun" to

start tasks. However, this approach still relies on ALPS and therefore has the same limitations. When the Agent runs within the cluster, only the initial startup of the Agent relies on ALPS. After that, all task launching is done within the cluster, without further interaction with ALPS.

3.3 Open Run-Time Environment (OpenRTE)

The Open Run-Time Environment (OpenRTE) is a spin-off from the Open-MPI project and is a critical component of the OpenMPI implementation. It was developed to support distributed high-performance computing applications operating in a heterogeneous environment. The system transparently provides support for interprocess communication, resource discovery and allocation, and process launch across a variety of platforms. OpenRTE provides a mechanism similar to the Pilot concept - it allows the user to create a "dynamic virtual machine" (DVM) that spans multiple nodes. In regular OpenMPI usage the lifetime of the DVM is that of the application, but the DVM can also be made persistent and we rely on this particular feature for RP.

Command Line Interface (CLI) Recently OpenRTE has been extended with tools to expose the creation of the persistent DVM ("orte-dvm") and the launching of tasks onto that DVM ("orte-submit"). This means that the setup of the DVM is a single ALPS interaction and that all task execution is then outside of the realm of ALPS. As RP is a Python application and OpenRTE is implemented in C, we choose to interface the two using the OpenRTE CLI. While this enabled us to push the envelope (i.e. 1000 concurrent tasks and support the sharing of nodes between tasks) we do run into new bottlenecks. As every task requires the execution of "orte-submit", the interaction with the filesystem is a limiting factor for task execution. In addition, as every task requires a "orte-submit" instance that communicates independently with the "orte-dvm" we also run into network socket race conditions and system resource limits with workloads that consists of large numbers of concurrent tasks. Although RP has the ability to spread the execution of tasks over multiple sub-agents (running on separate compute nodes), there is also a large centralized process footprint for maintaining state about each running process this way.

C Foreign Function Interface for Python (CFFI) CFFI provides a convenient and reliable way to call compiled C code from Python using interface declarations written in C. From an OpenRTE perspective this mode of operation is similar to the CLI mode but differs in the way RP interfaces with OpenRTE. Instead of running a tool for every task to launch it only requires a library call. This also allows us to re-use the network socket. The incentive for developing this approach was to overcome the limits in the CLI approach and prelimanary results seem to acknowledge this.

4 Experiments

In the full paper we will provide performance data comparing and contrasting different implementations for various synthetic and real-world workloads. The focus will be on the scalability and resource efficiency using the combination of RP and OpenRTE. The primary resource for the experiments will be Blue Waters , the XE6/XK7 system at NCSA.

5 Discussion

There is no "one size fits all" approach when it comes to HPC. While Cray systems excel at executing certain monolithic workloads, they are limited in running more varied workloads. RP in combination with OpenRTE is a non-invasive userspace approach that enables the execution of workloads that exceeds the design objectives of Cray systems. Using RP on Crays we have overcome many of the task execution limitations. We are admittedly trading-off some raw per-task performance as currently we can't run applications that are linked against the Cray MPI libraries. In the pre-OpenRTE era of RP, there was overlap with OpenRTE functionality. By leveraging the functionality of OpenRTE, RP can focus on the functionality that complements OpenRTE: the orchestration of tasks.