XCPU3 Workload Distribution and Aggregation

Pravin Shinde
Vrije Universiteit, Amsterdam
pse220@few.vu.nl

Eric Van Hensbergen
IBM Research Austin
bergevan@us.ibm.com

The mainstream adoption of cluster, grid, and most recently cloud computing models have broadened the applicability of parallel programming from scientific communities to the business domain. Despite the popularity of these emerging models, deploying parallel applications to these loosely coupled environments still requires substantial expertise in part due to the complexity of interactions between existing distributed middleware and runtime packages. We feel it is past time to re-evaluate a more tightly integrated model where distributed execution, process control, and communication are provided by the underlying operating system in a programming language and runtime independent fashion.

The concept of tightly integrating the facilities of parallel processing within the operating system is not new. Shortly after the availability of low-cost networks in the 70's and 80's, a number of distributed operating systems such as Ameoba [7], V [1], and others appeared from various academic research groups. In addition to implementing distributed file systems, authentication, and other services which remain popular today, they also provided tightly integrated mechanisms for remote process execution and control, allowing the resources of the entire network to be utilized as if it were a single system.

A slightly different approach was taken by the Plan 9 [8] operating system which provided a cohesive model for addressing and organizing a wide range of distributed resources through the use of synthetic file systems and explicit user manipulation of dynamic private namespaces. The explicit nature of the sharing and organization allowed the user to adjust the balance of the local versus remote resource utilization based on a particular workload, available network bandwidth, or resource availability. The downfall of these research systems was their inability to keep up in the application space as compared to mainstream UNIX.

The XCPU project [6] applied the Plan 9 mechanisms of distributed computing to high-performance computing applications running on mainstream operating environments such as Linux. It augmented Plan 9's remote execution model with a mechanism allowing the fanning out of many copies of a particular application to a set of compute nodes. XCPU would automatically determine an application's dependencies and push the application and associated shared libraries to a cache on the remote compute nodes. It used a novel tree-spawn mechanism to delegate control and distribution of application binaries, data, and dependencies to a subset of the compute nodes effectively instantiating an aggregation hierarchy allowing the mechanism to scale to thousands of nodes. It also used this hierarchy to broadcast input from the headnode to all participating nodes, and collesced resulting output back to the headnode.

A follow on project, XCPU2 [5], provided mechanisms giving users great control over the environment of the compute nodes. It allowed a file system view to be constructed per job which could pull resources from both the local node as well as a variety of network nodes and compose them into a private namespace. This combined the benefits of the XCPU distributed caching and tree-spawn mechanisms with the isolation properties of containers to provide multiple protected namespaces on compute nodes which could simultaneously run workloads from different distributions without concerns about cross-contamination.

XCPU2 's execution model was top-down with a single controller node and no ability of compute nodes to initiate subsequent computation. While XCPU2 utilized a synthetic file system interface to initiate and coordinate execution on the compute nodes, it used a traditional application on the client side to initiate provisioning and execution. Because of this, I/O aggregation was limited to a simple one-to-many input and many-to-one output aggregation model. As such, XCPU2 worked very well for most high-performance applications, but was quite limiting for commercial or data-flow workloads.

XCPU3 incorporates the scalable design elements of its predecessors while seeking to broaden the applicability of its approach to a larger set of high-performance as well as commercial application environments. It integrates the interface for job initiation into the synthetic file system, so that every node can act as server and client. XCPU3 also augments the aggregated I/O and control mechanisms with granular I/O and control interfaces allowing end-user and applications to leverage the most appropriate model. Finally, it adds an interface allowing the creation and connection of communication channels between threads regardless of their location within the network. The remainder of this abstract describes these mechanisms in more detail.

The XCPU3 interface is based around the Plan 9 model of providing synthetic file systems to define interfaces to devices and system services. Similar to the previous instances of XCPU, every node presents a set of top level files which provide information about the underlying architecture and system status. There is also a provisioning file, which can be used to instantiate a new task on the node. Active tasks are represented as subdirectories, each with interfaces for con-

Eurosys April 14–16, 2010, Paris, France.
ACM X-XXXXX-XX-X/XX/XX.

trol, standard I/O, and querying various elements of status.

All the XCPU versions use the 9P distributed resource protocol to provide the synthetic file system, allowing it to be accessed by remote nodes as well as mounted locally via v9fs [4] on Linux or FUSE on other platforms. Once mounted, any user application can interact with it directly. So, even though XCPU^3 is essentially middleware, it provides a system-level interface via the file system. This makes it extremely easy to construct a language binding for the mechanism, or even just interact with it directly from a variety of shell script environments.

XCPU^3 extends the use of the synthetic file hierarchy interface to allow initiation of execution on a set of remote nodes. This is accomplished through a new set of control messages which facilitate the reservation of remote resources. When using the reservation operations, the local task directory becomes the aggregation control point for the group of tasks replacing the need for the specialized client application or control node from XCPU and XCPU^2 .

This also allows any node within the cluster to initiate new sets of tasks elsewhere, turning the top-down hierarchy of XCPU into a peer-driven acyclic graph. This graph is automatically created and maintained in a decentralized fashion where each node makes decisions based on its knoweldge of neighboring nodes. Localizing the scheduling decision also allows XCPU^3 to use smarter algorithms which base decisions on this local knowledge. Any link or node failure can be dealt with at a local level without requiring reconstruction of the entire tree or restart of the workload. In typical large scale machines and data centers, certains nodes form natural aggregation points which can be utilized to make monitoring as well as workload reservation and deployment more scalable.

Previous version of XCPU only allowed a simple one to many control and I/O aggregation mechanism. This prevented granular control of any individual node. XCPU^3 adds flexibility by creating subdirectories within each task group representing each remote task. Interfaces within these subdirectories can be used to individually control a remote task or interact with only it's I/O. The parent's directory interfaces can still be used to interact with the set of tasks as a group, allowing the use of either interface or switching between the two as appropriate. This change also has the property of separating resource reservation from task execution, enabling the construction of complex dataflow pipelines of multiple component applications.

Within large scale deployments, such a mechanism could quickly get out of hand, with potentially hundreds of thousands of subdirectories for a large-scale, high-performance computing task. To prevent such problems, we build aggregation into our resource reservation and scheduling policy, forcing larger tasks to aggregate themselves by using the task interface recursively. As the processing for I/O aggregation is done at internal nodes and not only at a root node, it naturally increases the scalability by reducing the load on root nodes.

Many parallel applications require that compute nodes communicate with each other in order to directly share information for solving the problem. The traditional approach to this has involved enumerating the tasks participating in a particular computation and then using *MPI* or *network programming* to communicate between the various elements.

We wanted to explore providing communication channels which would enable emerging web based workloads such as MapReduce [2] as well as more dataflow oriented workloads such as DryadLINQ [10] or PUSH [3]. XCPU^3 adds interfaces to tasks allowing the creation and manipulation of *network file descriptors* which can be spliced together via control interfaces to provide one-to-one communication channels between tasks. These can be used together with helper tasks to compose more complicated communication pipelines involving fan-out, reductions, and multicast. Nodes can directly read and write into these network file descriptors using common filesystem operations like read and write. This interface simplifies the development of parallel application by replacing complicated network programming with simple file handling.

XCPU^3 is an attempt to apply distributed operating system principles to more conventional environments allowing easy interoperability with existing applications. It is open source and currently supports Linux, BSD, MacOSX, as well as the Plan 9 operating systems. We are currently using it to deploy a variety of workloads to thousands of cores on a Blue Gene/P as part of the HARE [9] project supported by the Department of Energy under Aware Number DE-FG02-08ER25851. More information on HARE and is various components (including this one), can be seen on our webpage[1]. A demo will be provided during the poster session, space permitting.

1. REFERENCES

[1] D. R. Cheriton. The V distributed system. *Communications of the Association of Computing Machinery*, 31(3):314–333, March 1988.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[3] N. Evans and E. Van Hensbergen. Push: a DISC shell. 2009.

[4] E. Van Hensbergen and R. Minnich. Grave robbers from outer space using 9p2000 under linux. In *In Freenix Annual Conference*, pages 83–94, 2005.

[5] L. Ionkov and E. Van Hensbergen. Xcpu2: Distributed seemless desktop extension. 2009.

[6] R. Minnich and A. Mirtchovski. Xcpu: a new, 9p-based, process management system for clusters and grids. *Cluster Computing, 2006 IEEE International Conference on*, pages 1–10, 2006.

[7] S. J. Mullender, C. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Stavern. Amoeba: a distributed operating system for the 1990s. 23(5):44–53, May 1990.

[8] R. Pike et al. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[9] E. Van Hensbergen, C. Forsyth, J. McKie, and R. Minnich. Holistic aggregate resource environment. *SIGOPS Oper. Syst. Rev.*, 42(1):85–91, 2008.

[10] Y. Yu et al. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Symposium on Operating System Design and Implementation (OSDI), San Diego, CA, December*, pages 8–10, 2008.

[1] http://www.research.ibm.com/hare