Putting A "Big-Data" Platform to Good Use: Training Kinect

Mihai Budiu
Microsoft Research SVC
Mihai.Budiu@microsoft.com

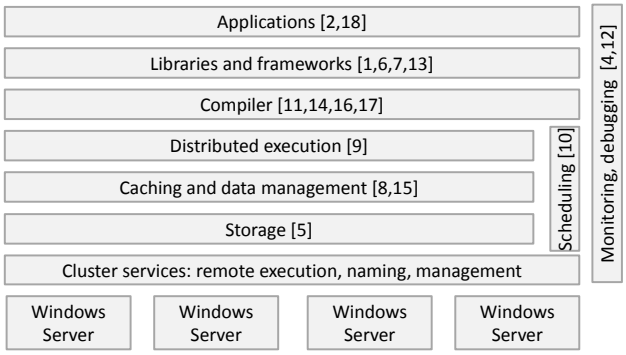

Figure 1: DryadLINQ software stack.

Categories and Subject Descriptors

C.5.m [**Computer Systems Organization**]: Miscellaneous

Keywords

Big data, LINQ, DryadLINQ, parallel computing, Kinect

1. THE DRYADLINQ PLATFORM

In the last 7 years at Microsoft Research in Silicon Valley we have constructed the DryadLINQ software stack for large-scale data-parallel cluster computations. The architecture of the ensemble is depicted in Figure 1. The goal of the DryadLINQ project is to make writing parallel programs manipulating large amounts of data (terabytes to petabytes) as easy as programming a single machine. DryadLINQ is a batch computation model[1], optimized for throughput; since it is targets large clusters of commodity computers fault-tolerance is a primary concern. A primary tenet is that moving computation close to the data is much cheaper than moving the data itself. Here we discuss briefly the current architecture of the system (but more research is ongoing).

Our software runs on relatively inexpensive computer clusters, using unmodified Windows Server. Our software makes minimal assumptions about the underlying cluster, and has been ported to several platforms: the Cosmos cluster runtime (built and used internally by Bing), Windows HPC Server and Windows Azure.

We have built a simple distributed filesystem, **TidyFS** [5][2]. All cluster machines have local disks, providing persistent storage; these disks are aggregated by TidyFS into a global filesystem. We assume that files can be very large, append-only, and thus partitioned into multiple pieces, each piece being replicated on several machines for fault-tolerance. Essentially, TidyFS is a reliable metadata service, allowing cluster applications to locate the actual data, which is stored as raw files on the local filesystem of each machine.

Dryad [9] is a distributed execution engine. Dryad programs (jobs) are coarse-grain, acyclic dataflow graphs. Each *vertex* in the job graph is an arbitrary computation, and vertices communicate with each other using point-to-point *channels*. The job graph is "virtualized," i.e., it is not tied to the available resources, and it can be executed by time-sharing these resources for the job vertices. For providing application-independent fault-tolerance Dryad assumes that job vertices are idempotent, computing a deterministic function of the contents of their input channels. Each channel contains a finite set of abstract records. Dryad is currently deployed by Bing for large-scale analytics, using many tens of thousands of machines and analyzing tens of petabytes of data every day.

In practice all Dryad computations are synthesized automatically from high-level languages. **DryadLINQ** [17, 11] is our compiler[3]. The input language of DryadLINQ is .Net. DryadLINQ takes advantage of the LINQ (Language-Integrated Query) extensions to .Net to generate parallel computations. LINQ is a small declarative and functional language embedded in .Net that operates on collections of values. LINQ is similar to the database relational algebra, but has a much richer data model, since LINQ collections can contain arbitrary .Net objects. LINQ computations are compiled into dataflow graphs using sophisticated optimizations [14, 16]. DryadLINQ takes advantage of the strong typing of .Net to generate efficient serialization code for doing data I/O. DryadLINQ not only parallelizes the computation across machines, it also parallelizes the computation on each machine across CPU cores. By leveraging the native support for LINQ in the Visual Studio IDE, DryadLINQ

[1]Our system does not address interactive computation, wide-area (grid) computing, in-memory computation (e.g., HPC) or streaming.

HPDC'12, June 18–22, 2012, Delft, The Netherlands.
ACM 978-1-4503-0805-2/12/06.

[2]DryadLINQ also interoperates with several other storage layers, such as Cosmos, the Distributed Storage Catalog, SQL Server, and even raw files exposed through remote file access protocols.

[3]Other compilers that target Dryad exist, such as Scope [3].

provides a seamless experience writing programs that span multiple jobs (workflows).

Many large-scale computations are performed on data sets that grow constantly (e.g., search logs). **DryadInc** [15] rewrites such computations to be incremental by rewriting Dryad graphs. the incrementalization effort can be automated with knowledge about program semantics, as done by the **Nectar** [8] system, which unifies storage, computation and incrementalization. Storage is treated as a cache of computation results, enabling a simple trade-off between (re)computation and persisting (partial) results.

In batch computing systems big jobs can impede the timely progress of other computations. The **Quincy** [10] scheduler attempts to provide a compromise between fairness and efficiency by optimizing resource assignment using a distributed max-flow model. Quincy takes advantage of the fault-tolerance mechanisms of Dryad to perform preemptive vertex-level scheduling, dramatically improving the interactive user experience of sharing a large cluster.

While DryadLINQ provides the illusion of programming a computer cluster as a single machine, the presence of bugs and failures may reveal the complexity of the underlying system. We have built tools such as **Artemis** [4] for profiling and visualizing the performance of distributed computations, and **Daphne** [12] for job introspection and debugging.

On these foundations we have built various domain-specific libraries: e.g., branch-and-bound search in **DryadOpt** [1], information retrieval [6], machine learning [7] and **PINQ** for privacy-preserving computations [13].

2. TRAINING KINECT

The reliability and usability of DryadLINQ were instrumental in overcoming some difficulties encountered when building the Kinect body tracking pipeline [2]. Kinect is a device designed to complement the Xbox 360 games console, enabling users to control the console using only body gestures and voice control. The hardware device provides a *depth* image, where each pixel is labeled with the distance to the sensor. The Kinect SDK computes a skeletal representation of the players in the device field of view, which is used by games and applications to control interaction. The most computationally-intensive stage of the Kinect body tracking pipeline recognizes and labels each pixel in the input scene with a (set of) likely body parts. This stage uses a decision forest classifier. This classifier has been built using supervised learning, by analyzing a large number of labeled images. Training the classifier requires a very large number of calculations, and has been implemented in DryadLINQ.

DryadLINQ provided several advantages compared with MPI: (1) a high-level parallel programming language for building distributed computations instead of a simple low-level messaging API, (2) automatic and efficient serialization of complex data types for transport on the network, (3) resource virtualization, which allowed the manipulation of datasets much larger than the collective memory of the available machines, (4) built-in fault-tolerance and checkpointing, and (5) tight integration with C#, which enabled us to reuse the libraries developed by the computer vision experts, and orchestrate the many parallel and sequential phases of the training in a single program.

3. REFERENCES

[1] M. Budiu, D. Delling, and R. Werneck. DryadOpt: Branch-and-bound on distributed data-parallel execution engines. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2011.

[2] M. Budiu, J. Shotton, D. G. Murray, and M. Finocchio. Parallelizing the training of the Kinect body parts labeling algorithm. In *Big Learning: Algorithms, Systems and Tools for Learning at Scale*, 2011.

[3] R. Chaiken, P. L. Bob Jenkins, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and efficient parallel processing of massive data sets. In *VLDB*, 2008.

[4] G. F. Creţu-Ciocârlie, M. Budiu, and M. Goldszmidt. Hunting for problems with Artemis. In *USENIX Workshop on the Analysis of System Logs*, 2008.

[5] D. Fetterly, M. Haridasan, M. Isard, and S. Sundararaman. TidyFS: a simple and small distributed file system. In *USENIX Annual Technical Conference*, 2011.

[6] D. Fetterly and F. McSherry. A data-parallel toolkit for information retrieval. In *SIGIR Conference*, 2010.

[7] D. Fetterly, Y. Yu, M. Budiu, F. McSherry, and M. Isard. *Scaling Up Machine Learning*, chapter Large-scale Machine Learning using DryadLINQ. Cambridge University Press, 2010.

[8] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang. Nectar: Automatic management of data and computation in datacenters. In *Symposium on Operating System Design and Implementation (OSDI)*, 2010.

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *European Conference on Computer Systems (EuroSys)*, 2007.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SOSP*, 2009.

[11] M. Isard and Y. Yu. Distributed data-parallel computing using a high-level programming language. In *SIGMOD Conference*, 2009.

[12] V. Jagannath, Z. Yin, and M. Budiu. Monitoring and debugging dryadlinq applications with Daphne. In *Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, 2011.

[13] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD Conference*, 2009.

[14] D. G. Murray, M. Isard, and Y. Yu. Steno: automatic optimization of declarative queries. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[15] L. Popa, M. Budiu, Y. Yu, and M. Isard. DryadInc: Reusing work in large-scale computations. In *HotCloud*, 2009.

[16] Y. Yu, P. K. Gunda, and M. Isard. Distributed aggregation for data-parallel computing: interfaces and implementations. In *SOSP*, 2009.

[17] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Symposium on Operating System Design and Implementation (OSDI)*, 2008.

[18] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, J. Currey, F. McSherry, and K. Achan. Some sample programs written in DryadLINQ. Technical Report MSR-TR-2008-74, Microsoft Research, 2008.