Review: Dryad

Louis Rabiet

September 20, 2013

What problem did the paper address? Who is the intended audience?

The paper is proposing to solve the problem of being able to take advantages of distributed programming without the difficulty usually associated with it. Some of the main challenges that you can encounter in distributed/parallel programming are:

- Management of failures: With the number of servers/nodes increasing and with unreliable connection (network, bus), failure is unavoidable.
- Handling consistency with several nodes/threads: Thread programming is known to be really challenging and expert domain only.
- Debugging distributed executions is also an hard problem.

Authors are using a well known technique (already present in Paralex[1] for example) to represent the scheduling of tasks using a graph. They are restraining themselves to a acyclic graph to better handle failure cases.

As their goal is to permit a non-expert (in parallel programming) to be able to develop an application in few weeks, they voluntarily hide the complexity of threads and concurrent programming.

Is it important/interesting? What was the context for the paper?

Their work is related to others pre-existing systems (at the time of submission) paradigm like Google MapReduce[2]. Compared to this system, Dryad's authors are claiming that they are able to to handle more general cases and to keep a good performance (without scarifying too much of simplicity).

Besides allowing several inputs/outputs sets (to compare with inputs set $\rightarrow$ map $\rightarrow$ computing $\rightarrow$ reducing $\rightarrow$ outputs sets of MapReduce), Dryad allow runtime optimizations (apparently not meant for user-side) such as aggregation (if several computations are similar, you can modify the graph to add new nodes to take advantage of locality for example.).

Reliability are basically the same than for MapReduce for the jobs: The job manager is a single point of failure and if a node (vertex) is failing then some part of the job is restarted (depending of whether the input was involved or not , more than one Venice can be restarted.)

What is the approach used to solve the problem?

The approach offered by Dryad is pretty simple, you need to represent the flow of execution of programs in the form of a graph (edges represent communication and vertices represent sequential execution).

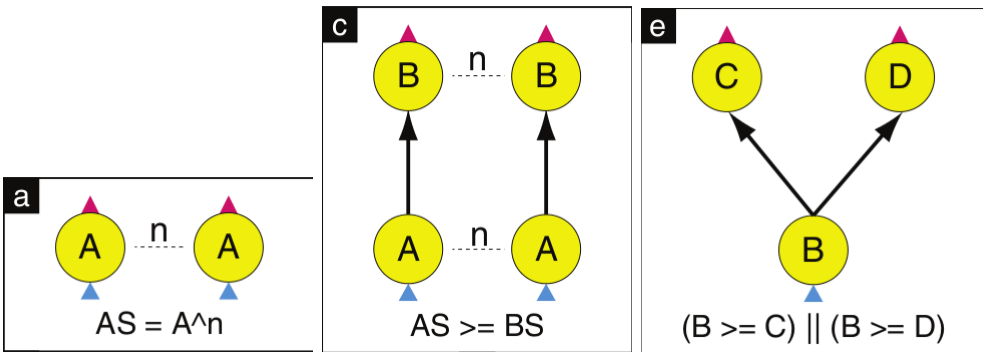

Figure 1: Some of graph operators from [4]

In the figure 1, a represents the possibility to divide the work in n similar parts, c represents the fact that the vertice A_1 need to be executed before B_1 and e represents the merge of two graphs $B >= C$ and $B >= D$. Compositions are modifiable and extensible by the users.

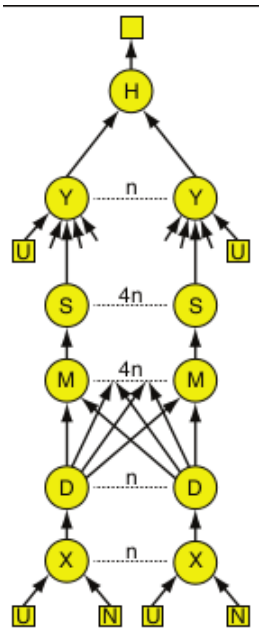

Figure 2: Simple program

In figure 2, the graph represented is the following: X and Y vertices are join of some databases. D, M and S vertices are the node that are used to do the actual computation, sorting (the actual goal is to find the neighbors of objects).

In figure 3, it is basically the $C++$ code describing the graph shown in the previous figure.

```
GraphBuilder XSet = moduleX^N;
GraphBuilder DSet = moduleD^N;
GraphBuilder MSet = moduleM^(N*4);
GraphBuilder SSet = moduleS^(N*4);
GraphBuilder YSet = moduleY^N;
GraphBuilder HSet = moduleH^1;

GraphBuilder XInputs = (ugriz1 >= XSet) || (neighbor >= XSet);
GraphBuilder YInputs = ugriz2 >= YSet;
GraphBuilder XToY = XSet >= DSet >> MSet >= SSet;
for (i = 0; i < N*4; ++i)
{
  XToY = XToY || (SSet.GetVertex(i) >= YSet.GetVertex(i/4));
}
GraphBuilder YToH = YSet >= HSet;
GraphBuilder HOutputs = HSet >= output;
GraphBuilder final = XInputs || YInputs || XToY || YToH || HOutputs;
```

Figure 3: Example of program done with Dryad's C++ API

What are the possible inefficiencies in this approach?

Such a distributed system need to restrain itself to a specific category of graphs to be able to schedule it. Here one the restriction is acyclic graph. Acyclic graph make it easy to have a partial order but if you allow the system to have some control over where a node is assigned you can have some equivalency between an acyclic and a cyclic graph.

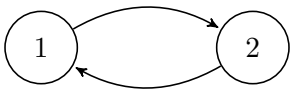

Figure 4: Cyclic graph

The Figure 4 can be transformed to Figure 5 and the two are equivalent if the system allow to impose constraints on the scheduler. But to be able to linearize the graph you need to have a way

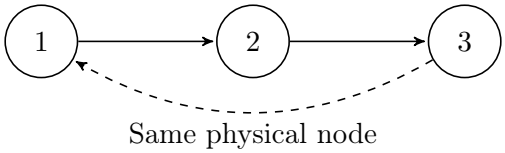

Figure 5: Acyclic graph

of specifying the number of occurrences of loop (if there is a infinity of exchange between A and B, the graph cannot be finite if we want to linearize).

One of the main deficiency of Dryad is the weak scheduler. A typical scheduler should be aware of data locality for example. To perform better in term of throughput, someone could implement something like [5].

How does the paper support or otherwise justify the conclusions it reaches?

To show that Dryad is performing well in term of performance (it is hard to know if the model proposed by Dryad will be widely adopted), authors have two experiments to validate their approach in Dryad.

- One experiment using SQL queries, they use the speedup (using a single SQLserver 2005 as reference) in two usage of Dryad: one in Memory one in Two-Passes mode. They are achieving almost linear speedup for Two-Passes (In-Memory is without surprise quicker) but that can be explain for several reasons
  - Dryad is only a light wrapper (execution engine) and most of the features/speed is provided by the SQL-server.
  - Locality is strong in the benchmark: Dryad is able to take advantages with runtime aggregation.
- The second one is a data mining test: the main interesting result is not the time needed for the computation but the correctness of the computation (with very large number of data (˜10 Terabytes).

What problems are explicitly or implicitly left as future research questions?

One obvious work left for the future is to integrate Dryad with a queries system like SQL or LINQ (stand for Language-Integrated Query and it is a language developed by Microsoft like Dryad and aim to bring query capabilities to language like C#). In 2008, the same group of researchers has developed the DryadLINQ[6] system. DraydLINQ is able from sequential code to transform it to a Parallel program that will be exploited in Dryad. In 2011, Microsoft announced that the DraydLINQ were left in favor of Hadoop (MapReduce from Apache Foundation). Although it has been shown that DryadLINQ can be compared in term of performances with Hadoop (in [3] for example), several features could explain the non-viability of commercial project as DryadLINQ.

- Globally the cost of licensing both Windows servers (DryadLINQ was meant for Windows servers) and DryadLINQ (Proprietary and commercial) compared to Unix servers and Hadoop (Free software developed by Apache) is a significantly higher.
- From a technically point of view, one of the main disadvantage compared to Hadoop is the lack of a real distributed file system. In short, in order to support large inputs, Dryad need to create a graph for virtual nodes and the communication is done via local write/distant read (File is the default channel protocol). In contrary, Hadoop is able to create high availability HDFS (for Hadoop Distributed File System).

References

[1] Özalp Babaoglu, Alessandro Amoroso, Luigi Alberto Giachini, Ozalp Babao Glu, Lorenzo Alvisi, Lorenzo Alvisi, Renzo Davoli, and Renzo Davoli. Paralex: An environment for parallel programming in distributed systems, 1992.

[2] Jeffrey Dean, Sanjay Ghemawat, and Google Inc. Mapreduce: simplified data processing on large clusters. In *In OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation*. USENIX Association, 2004.

[3] J. Ekanayake, T. Gunarathne, G. Fox, A.S. Balkir, C. Poulain, N. Araujo, and R. Barga. Dryadlinq for scientific analyses. In *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, pages 329 –336, dec. 2009.

[4] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[5] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. In *INFOCOM*, pages 1609–1617. IEEE, 2013.

[6] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.