The Case for Datacentric Grids

David Skillicorn
Queen's University
skill@cs.queensu.ca

OR

Why the ancient Egyptians are poor role models for designing grids

Facts about online data.

- It's big and growing fast
- It's naturally distributed
- It's (really, really) hard to move

1. Online data is growing rapidly.

Data stored online quadruples every 18 months – `data tombs'.

Disks being shipped growing at 112% p.a. (10 million TB shipped in 2000) – about the same rate as data growth, so there's no new `empty' storage out there.

Large web sites have web logs of 40GB/day (14.6TB/year).

Long distance carriers 350 million calls/day.

Processing power `only' doubles every 18 months.

2. Online data is naturally distributed.

Data is captured via multiple channels (customer data – web, phone, store; galaxies – different overlapping telescope technologies, different owners).

The cost/byte of storage systems grows rapidly beyond a certain size. Operating systems struggle to handle files larger than a few GB.

So it's not in data owners' interest to aggregate data in one place (e.g. a super news site), even though it might be in the interest of data users.

3. Online data is effectively immovable.

Pragmatics: few sites have enough swap space to handle the arrival of a terabyte dataset for temporary use.

Performance: there are some parts of the world with high available bandwidth; there are enough bottlenecks, however, that high effective bandwidth is unachievable across heterogeneous networks

Over global distances, latency is almost entirely time-of-flight and so will not be reduced by technology; this introduces real delays into computations (fudged by assuming preload).

Data has a property similar to inertia: it's easy and cheap to store, and easy and cheap to keep moving – but transitions between these two states are complicated and expensive.

Politics: data about individuals cannot be moved out of jurisdictions with strong privacy rules.

Data owners sometimes allow access to their data but not copying (mirroring).

Implications of datasets that are large, distributed, and immovable.

The moment when a dataset is processed is a very important moment – it may be the only time that data is ever looked at.

It's much more effective to divide programs into separate pieces and send them to the data. Moving data to the program sort of works today, but surely cannot scale.

This requires a datacentric view of computation, rather than the conventional processor-centric view.

1. A new programming model is needed.

   The `abstract machine' is different, so computations must be described in a new way. E.g. today, parallel programs assume that their data will be arranged in a convenient way. Datacentric programs must use data where they find it.

2. Applications must be decomposable.

   If a single logical dataset is partitioned (as it usually will be), each computation that accesses one of these partitions must be able to make progress locally; but these local results must be able to be combined into a single global result.

3. The results of (partial) computations must be small enough to move around.

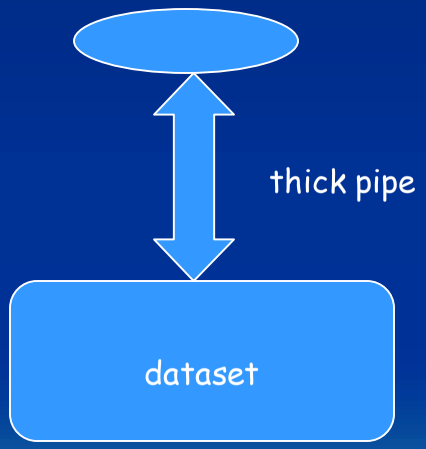
   Whatever a local computation does, it must produce condensed representations of its data. These might be models, statistics, or other samples.

4. These condensed forms are worth keeping.

   The extraction of condensed representations is expensive. Unlike other application domains, they are quite reusable. Hence, it is worth caching them. This requires additional storage, not only for datasets but also for their extracted models.

5. Execution nodes must be able to provide both compute cycles and high-performance data access.

Neither compute servers nor network-attached storage are of much use by themselves.

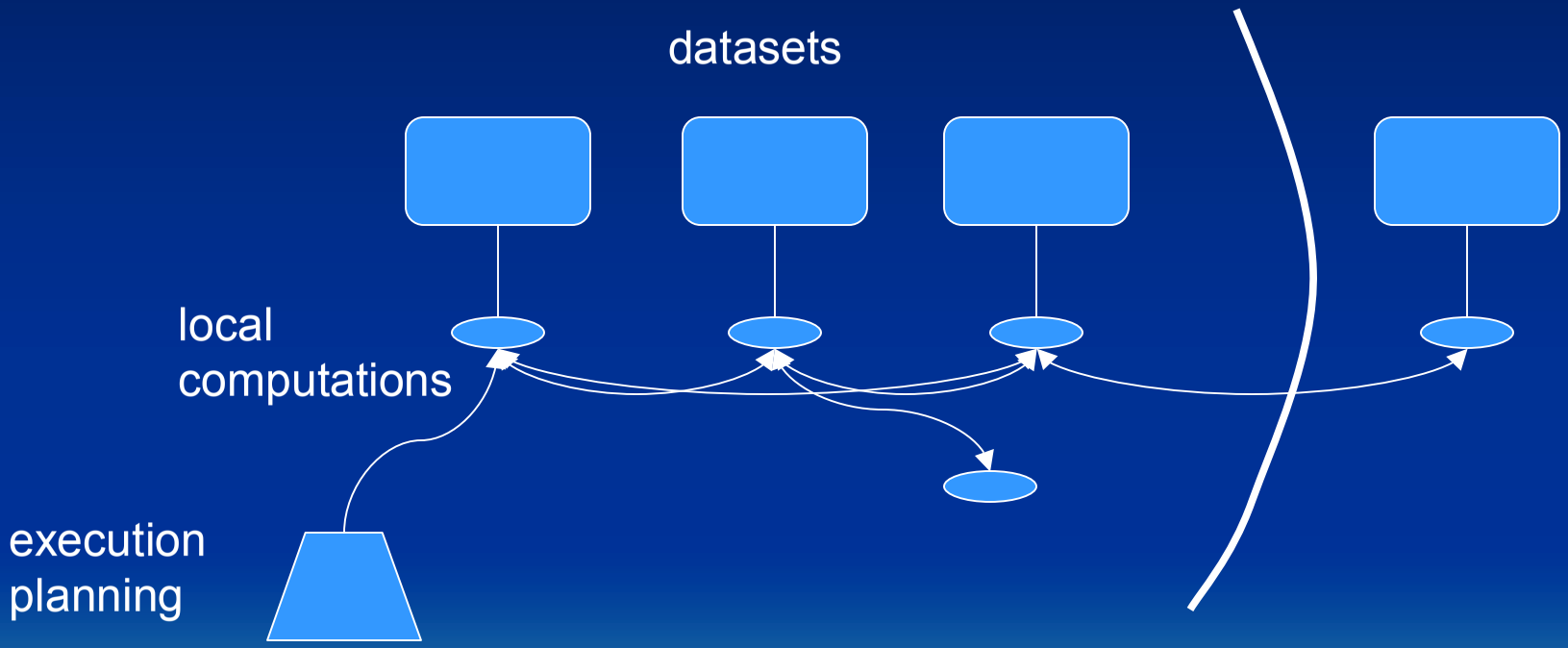


Some of the properties of datacentric applications that make them different from computational grid applications are:

- Users can be productive even from a thin client (whereas computational grids assume a fairly thick client)
- Applications require only thin pipes within the internet
- Code mobility is essential
- The format and content of a data repository will often be unknown to an application until it actually starts (locally) accessing it – a role for generic programming
- Applications will tend to be standardized (but run on user-specific datasets)

g. Applications will often be built from templates, perhaps even expressed using a query language
h. Re-execution of an application on a different or updated dataset will be common
i. There will be increased sensitivity about information leakage
j. As a result, applications are likely to run inside Virtual Private Grids (VPGs), but will need to access public resources without leaking (meta)information

Typical application structure.



The killer application for datacentric grids is parallel and distributed data mining.

Some examples:

1. Customer data collected at different touchpoints (web site, toll free number, store)

2. Using references in different online documents to measure interest in a topic

3. Analyzing galactic properties from multiple online catalogues

4. Investigating aircraft part failures by examining maintenance history in worldwide facilities

Many applications fit this distributed data mining framework.

But there are other types of applications that are also datacentric

1. Dynamic component-based application construction.

   The `data' are components themselves, and the task is to find the appropriate (and best-located) versions to assemble an application.

2. Mobile agent systems.

   When agents can only communicate locally (and so must actually mobility), they behave as datacentric computations.

Progress:

The framework for parallel data mining (IEEE Concurrency 1999) generalizes to datacentric grids for datasets divided `by row'. Merging techniques known for neural networks, inductive logic, bagging, arcing/boosting, frequent sets, SVD.

Some distributed techniques are known for datasets divided `by column' – Fourier coefficients, wavelets (Kargupta 1999-2001).

Mobile agents (with only local communication) – the Spider model.

Programming model? Modelling data repository contents? Storing hierarchies of models? New techniques for resource discovery and execution planning?

Existing Datacentric Grid Projects:

DCGrid at Queen's (Skillicorn, Huang, McConnell, Roumani), building on our experience with parallel programming models and high-performance data mining.

Knowledge Grid (Talia, Cannataro, Orlando), using Globus as a platform but developing components specialized for common data mining tasks.

Data Cutter can also be thought of as a kind of datacentric grid.

Other existing systems have datacentric aspects: Wikis, some kinds of web services.

www.cs.queensu.ca/home/skill/datacentric.html