Cassandra for Life Science

For decades, relational databases have ruled the world. For good reason, as this model effectively and consistently allows everyday information to be stored and retrieved by computer. The basic entity in a relational database is the table. Tables store information such as inventory items, customers, suppliers, addresses, orders, etc. These tables are then related to each other by one or more shared columns (or combinations of shared data). These relations allow for queries to be made on such databases, returning data sets that can be acted upon. For example, a list of all customers who have placed orders for a given inventory item within the last year might be contacted with a product update.

The success and ubiquity of the relational model has meant that it is the default model for new areas of application. Tried and true methodology is attractive when it becomes clear that computerized data management is required. Science is no exception, and the relational model has been applied with great success to most areas by now. Tables containing descriptions and properties are related to cover a particular domain of knowledge. The bulk of the work is in organizing data into ‘schema’ and translating data from previous stores (often very homogeneous, static, ‘flat’ data).

Life, however, is an entirely different proposition. In life sciences, the data are rarely static and ‘flat’. They are instead, quite dynamic and hierarchical. This is true at every level of magnification. In taxonomy it’s kingdom, phylum, class, order, family, genus, species (even this hierarchy has some blurred edges and exceptions). In organisms it’s population, individual, organ, tissue, cell, organelle, etc. In genetics, it’s DNA, chromosome, gene, base pair (roughly).

The proper representation of life is not tabular, but associative. The structure of life is not relational, but hierarchical. Relation is a poor term that falls far short of capturing dynamic connections. Saying that a Hox gene is ‘related’ to a set of genes involved in embryo ontology is like saying that a conductor is ‘related’ to the players in an orchestra. Shoehorning life science into relational databases is a very lossy process.

A database system is basically a model of the real world. The more closely it models the real world, the better are the chances of correct analysis, simulation, and even discovery. Better mapping of the real world also makes it easier for the user to conceptualize the data. Behavioural changes are a good example. Proper mapping of populations can make behaviours such as migration or quorum sensing stand out that might otherwise be 'hidden in the tables'. Properly structured data is more like knowledge.

In scientific research, it is cumbersome to wait for or rely on database implementors and maintainers who are often separated from the project at hand. Changes to the database structure are best made on-the-fly by database users. The user should be able to specify at least the local hierarchical structure. This principle also applies to database queries. Occasionally, users should be able to invent or test ad hoc queries, with the caveat that performance may not be good due to design issues, as we will see later on.

Many projects now entail widely distributed data gathering. Biodiversity databases are a case in point. Rigid, top-down management is difficult to maintain (especially as the database grows in size and geographic distribution). It may also discourage local refinements. Flexibility in the knowledge base itself is what is needed.

Life science requires a different paradigm for knowledge representation.

Cassandra

Cassandra is a non-relational database. It was originally developed by Facebook for internal use, starting from Google's 'BigTable' approach. It has since been released as open source, and is now under the purview of the Apache Software Foundation. It is currently in use by Facebook, Digg, Twitter, and several large cloud computing services. However, it doesn't come close to the widespread use of relational databases, and probably never will. Being an open source project, market penetration is not really the goal. Cassandra uses a different philosophy, one much more applicable to the life sciences.

Cassandra is scalable, distributed, fault-tolerant, self-replicating, eventually consistent, schema-less, and intended for hierarchical, object databases. That's quite a mouthful. Let's look at these features in greater detail.

Scalability

Many databases begin on a single computer at a single location, and work well in that initial situation. Then they run into problems when the system tries to scale up to more data and more users. Some database systems require such a significant 'buy-in' in terms of expense, supported hardware, and design requirements, that they cannot scale down to a smaller environment. Cassandra runs perfectly well on a single, modest PC for development, and can scale up to the petabyte (one million gigabytes) data size range spread across thousands of separate machines. These machines can be widely distributed, providing virtually unlimited expansion capacity. More importantly, this scalability is automatic, requiring no redesign or reconfiguration. Plugging in another machine increases storage capacity and processing speed. This is due to its 'BigTable' architecture, built-in 'garbage collection', and several other design features. Scalability also relates to performance (speed of read/write operations).

Cassandra maintains data in sorted order. This causes a small performance hit at data entry time, which is usually not a problem for human data entry. The big payoff however is that data lookup is extremely fast, even on huge databases. A range query is a query that will return an unknown number of results (0-n). Implementing such queries efficiently on most systems can be tricky. The problem of efficient, non-trivial queries is closely related to 'load balancing'. Much attention has been, and is being, paid to this area by Cassandra developers. The goal is to have a new node find a busy place in the network, providing automatic load balancing and thus better scalability.

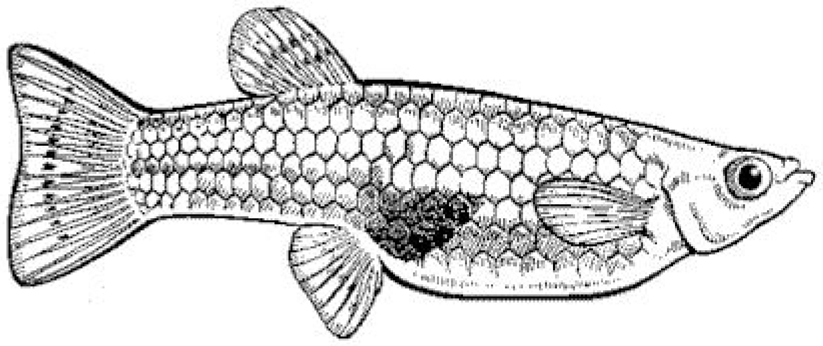
Scalability is an important consideration in life sciences. Genetic information is by its nature immense. Taxonomic and biological simulation data may start small, but will grow exponentially as the domain of study expands.

Fault-Tolerance

Computers fail. Power fails. Communications fail. The ability of a system to smoothly and automatically handle such failures makes it ‘fault-tolerant’. Cassandra implements a large data store as a cluster of nodes (again, which can be widely distributed). Data is automatically copied to multiple nodes (self-replication) to provide fault-tolerance. Failed nodes may cause reduced capacity, but will not cause the system to fail. Just as new nodes can be added at any time, failed nodes can be replaced without halting the system.

Another component of fault-tolerance is high availability. This means the system must have no single point of failure requiring a restart. Cassandra is ‘eventually consistent’, meaning that there’s considerable data caching going on in the background (as opposed to the strict transactional nature of relational databases). In other words, Cassandra is capable of running without stopping virtually permanently. This is important when a database is in use globally (24/7) and must be constantly available.

The distributed, decentralized nature of large Cassandra data stores provides a level of fault-tolerance in itself. Local problems, brown-outs, or bottlenecks can be mollified by the larger network. Trauma to individuals can be worked around by the colony.



Data Model

The most significant aspect of Cassandra is its data model – the way in which stored data is organized. It’s interesting to note that the basic data model Cassandra uses is not newer than relational database managers – it’s actually older, with its earliest lineage dating back to 1965.

In the early days of data management, the paradigm was structured storage. The relational model intentionally opted instead for much simpler, systematic tables in order to query the data (select and join) more easily (it’s difficult to pick out an ad hoc subset of a complex tree structure). That worked (and still does) for smaller scale data stores (perhaps up to around one terabyte). It also had advantages for strict transactional consistency, a requirement in business.

However, the relational model does not scale well, especially across multiple nodes. Chopping up tables is messy. Replication is therefore messy too. Easier, more automatic partitioning is causing renewed interest in older models because of the vast scale of some modern data stores.

Essentially, instead of a 'schema' comprising a group of related tables, Cassandra stores everything in a single associative array of unlimited size (unbounded). An associative array (also called a ‘hash’) is ultimately a set of name-value pairs, but in this case it’s actually a set of tuples (triplets) because a timestamp is stored along with the data to aid in conflict resolution among clients. Within this associative array, keys can map to multiple values, enabling a tree-like (or perhaps bush-like) structure. This is the view of life that a biologist has.

Here is the hierarchy, simplified, beginning at the top. A Cassandra instance is a container (cluster) of nodes (computers). Each instance is a container of keyspaces (databases). Each keyspace is a container of column families (files). Each column family is a container (ordered list) of columns. Columns are either simple or super. Simple columns store literal data (leaves). Super columns store sets of subcolumns (branches). Very much unlike a relational database, super columns (branches) can be added at any time. There is no ‘schema’ per se, because the actual structure of the database is dynamic. For example, an entire new branch of taxonomy can be added with a super column.

Much time is spent in normalization (removing duplication and empty space) when implementing relational databases. Improved performance requires that the schema comprise many small but related tables. Intelligent indexing is also a major design and implementation concern. The situation is made even worse when trying to normalize inherently hierarchical data. The database design can quickly become complex and inefficient.

In Cassandra, normalization is of lesser benefit due to the blinding speed of most database updates. Indexing is not automatic, open to innovative customization, and of lesser general concern (because the data is maintained in sorted order). In Cassandra, the structure of the database is really 'hard-wired' indexing. To implement efficient queries, intelligent design of column families and their associated keys is required (column families are implemented only at design time). Knowing which queries will be popular is important. In the Cassandra model, the design proceeds backwards from expected queries to organization and grouping of data. The need to compensate for badly placed and organized data with extensive indexing is largely removed.

Much of what has been said about indexing also applies to sorting. The columns (literal data) are stored in sorted order by the name in the name-value pairs. Sorting options (such as comparison rules) are generally specified at design time in Column Family definitions.

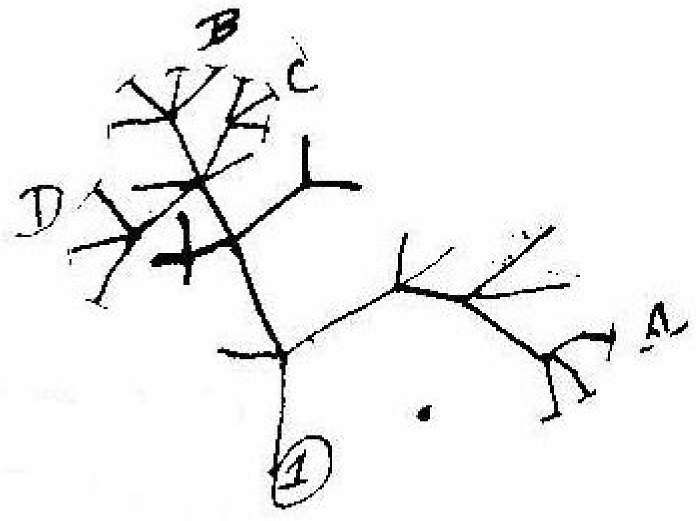
In summary then, the Cassandra model aims to greatly alleviate the burden of database maintenance by removing many of the contortions relational systems must endure to make unstructured data stores query-able. SQL comes with a hefty price tag in terms of both performance and maintainability. In fact, Cassandra-like databases are commonly known as 'NoSQL'.

Limitations

Cassandra is fairly new, so most limitations are only there because no one has got around to overcoming them yet. One example is the load balancing concern mentioned above. Another is Cassandra's general inability to handle large binary objects (such as video streams).

There are a few built-in limitations though. One is the requirement that each single key-column's data must fit entirely on one node. This is due to the way Cassandra implements self-replication, and generally poses no practical hindrance. Another is the necessity of synchronizing the clocks on all the clients due to the timestamping of data mechanism. Another is the use of a special key-value store to begin with. A different approach might be to implement the entire data store in XML. Another is that Cassandra does not support SQL (the predominant query language used in relational databases). As mentioned earlier, this is as much a feature as a limitation.

The Internet has now become the most standard platform for distributed applications such as scientific databases. One of the main languages for web development is Java. Cassandra is written in Java, which also can provide good access control and security. Theoretically, any operating system that supports Java can run Cassandra, but in practical terms, Linux is the only one supported.



Conclusion

With relational databases, the structure is rigid while the usage is flexible and extensible. In Cassandra, it's largely the other way around. In short, Cassandra is best for predictable usage of an evolving knowledge base, which is not usually the case in business, but is almost always the case in life sciences such as cellular biology, genetics, cladistics, and biodiversity.

Open source software offers several advantages over proprietary, commercial packages. The obvious one is cost. Not only is open source 'free', but the user benefits from the efforts of many enthusiastic open source developers. There is no commitment to a commercial company whose fortunes and priorities may change without notice. Also, the source code is available and modifiable if needed.

If what you need a database for is less like "Widgets, Inc." and more like Google, Facebook, or Twitter, then give Cassandra a look.

Resources

John Quinn
"Saying Yes to NoSQL; Going Steady with Cassandra"
http://about.digg.com/blog/saying-yes-nosql-going-steady-cassandra
March 2010

Arin Sarkissian
"WTF is a SuperColumn?"
http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model
September 2009

Raphael R. Some and Akos Czikmantory
Jet Propulsion Laboratory, Caltech
"XML Hierarchical Database for Missions and Technologies"
IEEEAC paper #1001, September 2003