Networked Data Management Design Points

James Hamilton

Microsoft SQL Server Development
One Microsoft Way
Redmond, WA
USA
JamesRH@microsoft.com

1. Introduction

Data management in a networked world presents us with some of the same challenges that we've seen in the past, but emphasizes our ability to deal with scale, in that there are several orders of magnitude more database users, and database sizes are rising more quickly than Moore's law. We have considerably less control over the structure of the data than in the past and must efficiently operate over poorly or weakly specified schema.

After more than 30 years of evolution, and substantial commercial success, database technology is more relevant now than ever. We are close to being able to keep all human produced information stored and machine accessible, and to having all devices owned by all people online. These advances will force us towards new client device programming models, and will require much more research in query processing techniques and algorithms, not just on performance, but on returning approximate result sets, new indexing techniques, and support for a multi-tiered caching data management hierarchy. Database administration will move from a job classification to an automatic system performed operation, and there will be dramatic changes in database server architectures.

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

2. The Client DB Environment

International Data Corporation (www.idc.com) estimates the number of US Internet users in 1998 at 51 million and the worldwide count at 131 million. In 2001, they expect the worldwide count to hit 319 million, at which point there will be 515 million connected devices. And, although these estimates seem quite reasonable, the numbers could arguably be substantially higher once we start networking non-standard computing devices such as VCRs, televisions, and other home automation functions. Just surveying my house and counting devices that have memory or internal state that require setting or configuration produces over a hundred count including (often with multiple copies of each): cell phones, water heater, clocks, radios, sprinkler controllers, oven (clock), microwave (clock), watches, televisions, CD players, VCRs, PH controller (fresh water aquarium), wave maker (marine aquarium), aquarium light controllers, home thermostat, desktop computers, laptop computers, several palmtop computers, refrigerator, and many more. Each of these devices requires independent attention to be set, many need to be reset after a power failure, and none of these devices cooperate or feedback today. Just setting all the clocks in the house after a power failure can be a bit painful given that few modern devices don't include a clock these days (VCR, stove, microwave, TV, thermostat, etc.). Instead of having on average, 1 to 2 devices per person connected to the net, we could see several hundred each, yielding a 2 orders of magnitude increase in the number of network-connected devices ranging to upwards of hundreds of billions.

Connecting these devices and having them interoperate and share data is quite compelling. Rather than putting on my home air-conditioning when the house temperature starts to climb it might make sense for the system to

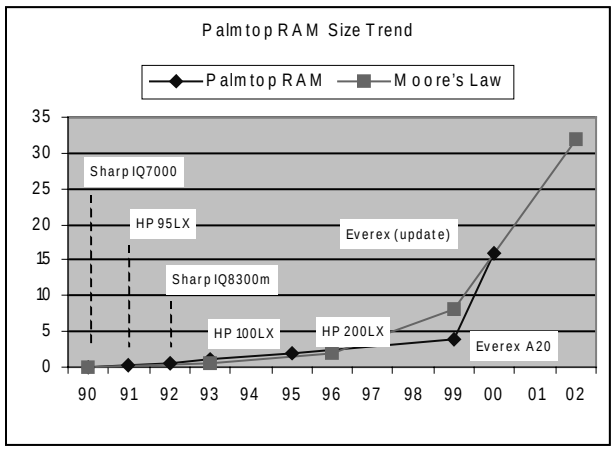
observe that I'm not currently in the house and shut off the 4,000 watts of lights over the fish tank and ensure the the tank still receives the appropriate daily photoperiod by putting them back on once the house has returned to a more moderate temperature.

Clearly there are many inhibitors to such interconnection happening in the near future, such as the multiple-hundred million non-networked install base, and the lack of a widely supported appliance interconnection standard. To be successful an existing infrastructure such as power line, radio, or telephone must be exploited rather than requiring that the home be rewired. So, although it won't happen immediately, it appears inevitable that it will happen. Microsoft and a consortium of others have proposed Universal Plug and Play (www.upnp.org) as an appliance interconnection standard. Sony is aggressively supporting IEEE 1394 as a digital appliance interconnection mechanism (www.sel.sony.com/semi/ieee1394wp.html) and they predict that the market for these devices will grow from the current 2 million devices to 460 million units by 2011.

The growth of the network-connected user population, in conjunction with a substantial predicted increase in the number of network-connected devices per user, will lead to 100's of billions of network connected devices. And, of course, the whole point behind network connecting these devices is so they can share data. In effect, each of these devices is a database client in a huge web of interconnected database management systems.

2.1. Is Client DB Size Really the Issue?

Currently, DB footprint has been the dominant issue in discussions of client side database management systems perhaps followed by the implementation language in which the database was written (SIGMOD99, "Honey I shrunk the database: footprint, Mobility, and beyond"). I argue that both features are either irrelevant or rapidly becoming so.



Consider the device resource growth driven by Moore's law. We are very close to having the resources even on the smallest devices to support standard applications and programming infrastructure. In the figure above, I plot the RAM sizes in black of the 5 palmtop devices I've owned over the last 10 years and, in grey , I plot the Moore's law predicted growth.

This data predicts that a commodity price point (under $500) palmtop device will have 32 MB of memory by 2002 and, if the current memory growth rate continues, will have 128 MB before 2005. Since most of these devices exhibit a significant degree of specialization and therefore have less than general system memory resource requirements, it would appear that we are very near to having the memory resources needed to support standard operating systems, database management systems, and applications. Special device specific programming models and database programming interfaces can only be justified when space is unaffordable. Even a year ago this was obviously the case. However, with available memory growing exponentially without increase in price, why would we trade off programmer time to rewrite features a second time in order to save memory resources? Programmer costs are rapidly increasing, and the limiting resource for most software projects is skilled developers. Even saving half of the memory in a more memory efficient implementation cannot justify or recover the additional time and expense of a custom infrastructure.

Saving memory at this cost is unaffordable.

The argument that the database implementation language is important centers around the need to automatically install the application software and database software infrastructure on each client when needed. Obviously this is an important feature but, equally obviously, Java isn't required to achieve this goal. For example, Office 2000 is composed of many components most of which don't need to be installed until needed. When a feature is accessed, it can be automatically installed on the client. As client-side devices evolve, all the data stored on client system should just be a cache of those most recently used applications and data. Nothing should have to be explicitly installed. This is an important attribute of a networked client database-programming infrastructure but it does not dictate that any particular programming language be used.

If database footprint and implementation language aren't the most important attributes of a networked data management infrastructure, then what is? I argue that the number one characteristic will be supporting a programming interface and execution environment that is symmetric with the server tiers. Any program should be able to run on any device, anywhere in the multi-tier computing hierarchy. Different devices will exhibit

different performance characteristics, many will be optimized to support a narrow set of features at a given point in time, but every device will support exactly the same programming interface and execution environment. Further, such infrastructures will support automatic installation of both data and applications, in effect just being redundant caches of resources available elsewhere on the network. The distinguishing features of client-side devices should be input/output device support, size, and battery life.

3. Networked DB Design points

3.1. Save Everything for all Time

In "How Much Information Is There In the World" (www.lesk.com/mlesk/ksg97/ksg.html), Michael Lesk argues that we are on the cusp of being able to store all information produced by the human race and never need to delete anything. His argument is based upon summing all the yearly media sales and information sources in the US and then extrapolating to worldwide quantities using GNP ratios:

- Paper sources: less 160 terabytes
- Cinema: less than 166 terabytes
- Images: 520,000 terabytes
- Broadcasting 80,000 terabytes
- Sound: 60 terabytes
- Telephony: 4,000,000 terabytes

These data yield a sum of approximately 5,000 petabytes. Others have estimated the number to be as high as 12,000 petabytes but most estimates appear to agree that the total amount of data produced annually is on the order of thousands of petabytes.

Looking at the worldwide storage production as estimated by Optitek for 1998, we get about 13,000 petabytes leading to the conclusion that annual tape and disk production are on verge of being able to store all human produced data.

At least three database management system related conclusions follow from this data: 1) the amount of online storage is climbing prodigiously fast and it's increasingly network attached, 2) the bulk of the data in the world is unstructured or only weakly structured, and 3) current query techniques and algorithms will perform poorly on a corpus with these composition and size characteristics. Much of the data that is currently without apparent structure will be self-describing (with XML being the most likely mechanism). Query systems will efficiently exploit hidden structure returning less "noise" on each query and allowing efficient joins between data in different data storage systems.

Pre-computed results, summary tables, materialized views, multiple redundant index structures, and detailed distribution and cardinality metadata are all space/query time trade-off that will make sense as our online data sizes rapidly increase.

3.2. DB Administration a Major Deployment Barrier

Server administrative costs dominate both software and hardware costs, and with hardware and software costs continuing to decline, this ratio worsens each year. However, some early research is showing significant promise. Surajit Chaudhuri of Microsoft Research is leading the Autoadmin project (http://www.research.microsoft.com/research/db/AutoAdmin/default.htm) the focus of which is to automate all database administrative activity. Early work on index tuning was published at SIGMOD'98. Essentially, auto administration techniques trade off machine resources to perform database administrative tasks. Given that hardware costs are rapidly trending towards zero while, database administration costs stubbornly remain constant or even climb year-to-year, this is an excellent trade-off. SQL Server 7.0 shipped the first result of this research, the index-tuning wizard. As research activity in this area increases across the industry, we should eventually be able to automatically create and destroy indexes as dictated by query load balanced against update activity and available system resources. Materialized views are another form of redundant data that today require administrative tuning but, over time, should be automatically managed by the system. Another automatic administrative feature demonstrated by Microsoft SQL Server is the automatic computation of data distribution and cardinality statistics as needed by the optimizer. Resource-based administrative decisions are much better made dynamically by the database management system than left to an administrator, partly to avoid the administrative costs, and partly because automatic reconfiguration adapts more quickly and efficiently to changing query loads. Microsoft SQL Server has implemented dynamic memory reconfiguration, and other features such as automatic degree-of-parallelism selection are common in commercial products. So, the industry as a whole is making progress, but it remains perhaps the most important problem we face.

3.3. Affordable Availability

The ubiquity of the Internet allows many companies to make their back office systems directly accessible to customers without sales people as intermediaries. This can dramatically decrease the cost of providing a service and improve the quality of that service, but it has the effect of making the back office system into the front office. There no longer is a layer of service or sales representative between the back office system and the customer, so any interruption of service has immediate

negative impact. In addition to this intolerance to unexpected downtime, 24-hour availability requirements make it very difficult to perform periodic offline maintenance and upgrade operations. Paradoxically, database management system size and complexity has been on the increase for years – many systems have grown by at least a factor of two over the last 5 years – at the same time that system availability is becoming increasingly important.

From spending many attempting to build bug free database management systems, it became increasingly clear to me that, as an industry, we do an excellent job of removing functional errors. The bugs that actually get missed tend to be complicated combinations of different operations and sequences of relatively unlikely events. These problems are insidiously difficult to isolate without years of testing and we simply can't wait that long prior to shipping. This class of bug is very difficult to remove but their interesting characteristic is that, if a failed operation is re-run, it's very unlikely that the same sequence of multi-user events will again lead to failure. Typically a failed operation will succeed if it is re-run. Jim Gray has described these bugs Heisenbugs in the talk "Heisenbugs: A Probabilistic Approach to availability" (http://research.microsoft.com/~gray/Talks) and in an earlier paper. Several commercial database management systems exploit characteristics of these bugs to increase system availability. For example, Tandem will attempt to re-run the operation on a surviving process pair and Microsoft Exchange will re-try failed I/O operations. We need to make systems resilient to such failures rather than trying to remove them at great time and development expense.

3.4. Redundant Data, Summary Data, and Metadata

Efficient point access to data, the core of most transaction processing systems, is very close to a solved problem across the industry. TP systems tend to scale with the number of user, number of customers or, in the limit, the number of people on the planet. However, large decision support and data analysis problems are not nearly so well behaved. Database size is growing faster than Moore's law (David Patterson; SIGMOD'98 keynote address) and the amount of data referenced by a single query can be a substantial part of the total database size. While many improvements remain to be made, a couple of factors are working in our favor. Perhaps the most significant is that dramatic increases in the amount of available storage online will allow use of multiple redundant copies of data for more efficient data access. These copies include, for example, mirrored disks where the non-busy member of the pair can support a read while the other member is busy. Also included in this class are index structures and materialized views, which allow query results or sub-results to be stored and automatically maintained. In addition to being able to store redundant copies of data affordably, we'll also continue to allocate larger percentages of the total database size to storing statistics about the data (data distribution and cardinality information), helping the query optimizer choose more efficient query plans.

We need to continue improving indexing and materialized view techniques, and devise other mechanisms to exploit multiple data copies in support of more efficient access. However, as we continue to increase the number of possibilities, we need to recognize that systems are becoming increasingly difficult to tune. These indexing structures and other metadata need to be automatically produced when needed and removed when their cost/benefit is no longer positive for a given systems work load. Leaving the decisions on which structures to add and when to remove them to administrators, as many current systems do, is contributing the scaling problem rather than helping to addressing it.

3.5. Data Structure Matters

Most Internet content is today unstructured text and, as a consequence, search techniques are restricted to simple Boolean search through unstructured text, with the side effect of the search not being particularly selective, and typically returning many irrelevant documents mixed in with the search targets. All of these documents actually do have some structure, but it's typically not explicit. Search products, such as Yahoo, go through and categorize the online corpus but as more and more data is put on-line, this manual approach won't scale. Further, the categorization provided by these search products is fairly limited, typically only categorizing at a high level and not parsing out and representing finer document structure.

There exist many possible solutions to this apparent lack of structure, but XML is emerging as a good mix of simplicity and potential richness to become the structure expression language of the Internet. Database management systems need to support XML data as a first class type and support efficient searching and processing of XML data.

3.6. Approximate answers quickly

Current database management systems have specialized for years in computing the correct answer when queries are posed, however, text search systems have shown for years that an approximate answer computed quite quickly can be more valuable than waiting for the right answer.

Joe Hellerstein has begun to exploit this observation by working on database query processing techniques that produce an approximate answer very quickly and then steadily improve the accuracy of the result. This is a

novel approach that is made far more useful by constantly showing both the current approximate answer and the statistical confidence bound on this result. Peter Haas and Hellerstein wrote up this approach and some incremental query processing techniques in "Ripple Joins for online Aggregation" (SIGMOD'99). Also, presented at SIGMOD'99 was "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets (Jeffrey Vitter and Min Wang).

I believe that these techniques have tremendous potential to fundamentally improve the efficiency of database users working on analytical problems. Clearly, there will always be a place for conventional algorithms that compute the exact answer as quickly as possible, but I can imagine using approximate techniques to explore a search space and form a hypothesis and then an exact technique to prove that an interesting data trend actually does exist.

3.7. Convergence of Data Processing and Data Storage

David Patterson in his SIGMOD 98 keynote address (cs.Berkeley.edu/~patterson/talks) made the case for intelligent disks arguing:

- I/O bus bandwidth is a bottleneck to delivering bandwidth
- Fast switched serial networks can support enormous bandwidth
- Processor/memory interface is a bottleneck to delivering bandwidth
- Growing CPU-DRAM performance gap leading to under-utilized CPUs

And, as a result, we have a new system opportunity in combining CPU, high-speed serial network, and disk together in a single package that, incidentally, was already required by the disk. In fact, the power and chassis requirements of these disks would be largely unaffected.

In the NASD presentation "Put Everything in Future (Disk) Controllers (It's not "if" it's "when?")" (www.research.microsoft.com/~gray/talks) Jim Gray agrees with Patterson's prediction that we will need to move processing power to data sources. Gray goes on to state that these devices should be leveraging commodity parts, including standard operating systems and standard database management systems, rather than special purpose real time kernels and purpose built software. His point is very similar to the one that I made earlier in this abstract when I argued that light-weight client devices (palmtops, etc.) should be built using standard applications, system software and data management infrastructure. Leveraging the existing programming environments and skills is much more important than optimizing for hardware utilization and exploitation.

I agree that programs, data management software, and standard operating system software will be implemented in cyberbricks and this conclusion argues strongly that database management systems will have to be built to support clusters of thousands, rather than existing design points of tens to, at a stretch, hundreds of nodes. And, rather than a cluster only being suitable for very large databases or very large numbers of users, it would become the standard component from which all database management systems would be constructed. The only difference between a small system and a large system would be the number of nodes employed.

4. Conclusion

Managing data in a networked world presents us with some of the same challenges that we've always faced – scale and robustness -- but on a scale several orders of magnitude beyond past experience. XML will bring some structure to what has appeared to be unstructured data and provide a common language for the expression of metadata about the documents and about the corpus as a whole. Even with these expected improvements in document structure, we're expecting huge changes in our handling of weakly structured or unstructured data and we will need to introduce far more schema flexibility to efficiently store and query this data. The sizes of our databases will have grown to the point where the focus has shifted from operational access to data analysis and data mining – some data items may, over the course of their storage lives, only be summarized or aggregated and never directly accessed. A statistically near answer quickly is rapidly becoming much more useful than the "right answer" and, more than ever, we will be trading space for performance in the storage of indexes, statistics, summary tables, etc.

Rather than supporting low end client devices with specially developed applications and data management infrastructure, decreasing device cost and stable to increasing programmer cost will force us to a model where the programming model and infrastructure software is the same across all tiers in the computing hierarchy.

Applications will continue to migrate to the data with servers composed of many cooperating cyberbricks each slice of which is a disk, CPU, memory, network connect component. And, data will continue to replicate to client devices where each client is a large cache of recently accessed data and recently used programs.