Rethinking Database System Architecture:

Towards a Self-tuning RISC-style Database System

Overview of the Problem

- Databases are widely popular and vital
- laden down with features
- increasingly complex
- pain of configuring/maintaining beginning to outweigh the gain of using it

Overview of the Goal

Databases should be:
- Easy to manage
- predictable in their performance characteristics
- self-tuning

Overview of the Solution

To achieve the aforementioned goals:
- RISC-style simplification of server functions
- small data managers with specialized APIs
- self-assessment/auto-tuning capabilities

The Case for Departure

- Expenses due to administration and tuning dominate the cost of ownership of a database system.
- Databases are packaged into a single unit of deployment, development, maintenance, and operation.
- large footprint, requiring database to exist separate from applications.

Incredible Crisis: Feature Creep

- "Featurism drives products beyond manageability."
- features added for marketing
- database systems become overloaded with features, when only a small fraction of those features are ever used.

Incredible Crisis: The Pain of SQL

- "way too complex for the typical application developer"
- the "core" is useful, but the bells and whistles gum up the works.
- SQL becomes highly unreadable when trying to put everything into one statement.

Incredible Crisis: Unpredictable Performance

- time-to-market pressures and feature creep lead to highly unpredictable behavior and performance.
- no one understands all the nuances of query optimization
- Unfortunately, service quality guarantees are more and more often a necessity.

Incredible Crisis: Tuning=Nightmare, Auto-Tuning=Vaporware

- Modern database systems, designed for a wide range of applications, provides scores of tuning knobs. These must be configured by gurus or fine-tuned through trial and error.
- Universal default settings simply don't exist, and auto-tuning is still in the research phase.

Incredible Crisis: Playing with the Neighbors

- Databases often used as glorified BLOB servers, due to complexity.
- Many applications add an additional layer of querying and query optimization so that the functionality can be tuned to a specific domain.

Incredible Crisis: Giant Feet!

- There is an emerging market for database systems to run on embedded systems, such as palm pilots and mobile phones.
- Unfortunately, the bloated system requirements make this a hefty task.

Incredible Crisis: Database Research Sucks

- The complexity of real world database systems makes research a frustrating prospect.
- systems-oriented database topics have been beaten to death.
- teaching databases is no fun, because of all the tricks and hacks found in some systems

It's a Trap!

- Universality-the trend to make systems general purpose, decreasing their usefulness when taken too far
- Cost- since it's cheap to 'manufacture' all the features get crammed into one system
- Transparency- high level functions hide expensive operations. You can't necessarily use the full expressiveness of SQL and expect the query optimizer to work magic.
- Resource Sharing- putting disparate applications (such as video streaming and traditional data sources) on a single system causes an extremely nasty tuning problem.

Previous Attempts: Database System Generators

- generates customized database systems from a large library of primitive components
- the subtle interplay of cache management, concurrency control, recovery, query optimization and other components makes generation of custom configurations difficult, if not impossible.
- interesting, but not successful

Previous Attempts: Extensible Kernel Systems

- put core funtionality into a kernel system and provide a means for extending the functionality.
- "data blades", "cartridges", "extenders"
- extensibility with regards to ADTs and UDFs is going well, but extending the internals is a nightmare, and pretty much impossible.
- most extensions written by the vendor anyways.

Previous Attempts: Unbundled Technology

- unbundling database systems and exploiting it in many varied services
- mail servers, document servers, switching and billing in telecommunication.
- close to RISC-style, but doesn't address the future of database systems as a discrete entity.

RISC Style Components- Requirements

Each component must:

- support richer components built on top of them (layered approach)
- be clearly separated from other components
- have a well defined and narrow functionality

RISC Style Components- Properties

- predictable behavior/self-tuning capability (due to simplicity).
- compartmentalized nature makes it suited for various applications.
- even monolithic systems benefit from component-oriented approach, due to reliability.

RISC Approach to Queries

- single table selection processor, which supports single-table selection processing and simple updates with B+ indexing.
- even this simple component is useful in many contexts
- supports a programmer-friendly API, due to its simplicity, and SQL can simply be ignored

RISC Approach to Queries

- Select-Project-Join (SPJ) query processing engine, suitable for OLTP
- built on top of the single table selection processor
- much more well understood than full-blown SQL engine
- adding in aggregation makes it quite powerful
- layering allows us to view aggregation as a problem in terms of SPJ query sub-trees.

RISC Approach to Queries

- full-fledged SQL on top of SPJ+Aggregation
- decomposes the optimization problem and thus the search complexity
- Reduced functionality and raw performance are the trade-off for reliability and predictability

Other Component Possibilities

- Storage Management, with different components for multimedia as opposed to generic data.
- index manager (immediate versus deferred index maintenance)
- tuning becomes a per-component operation, and the uncertainty of monolithic tuning disappears.

Ramifications

- componentization limits interaction among components- limited APIs are the only methods of communication
- API should expose functionality and import/export of meta information
- meta information would be used to gather performance estimates and influence query execution

challenges

specify the components such that:

1. the interfaces can be exploited by a number of applications
2. the performance loss is tolerable
3. each component is self-tunable and predictable

Simplification Recommendations

- support only for limited data types: limiting the responsibilities of a dbs to data in table format makes the system much more manageable
- no more SQL: instead, a streamlined API where programs submit operator trees to the database server modules. the key to simplification lies in limiting the functionality and expressiveness

Simplification Recommendations

- Disjoint, manageable resources: No dynamic resource sharing among components. Each major component should have its own hardware (video server, text document server, table manager, etc).
- Pre-configuration: each component should come pre-configured with 5 or 10 “power levels”. (basic, advanced, etc, or “mostly read workloads, small to medium data volumes, etc)

Prerequisites

- Universal Glue: OLE-DB or EJB or some such so that all components know how to talk to each other.
- Occam’s Razor: select the simple and necessary features when choosing/developing components.

Prerequisites: Self-Tuning Framework

1. Identifying the need for tuning
2. identifying the bottleneck
3. analyzing the bottleneck
4. estimating the performance impact of possible tuning options
5. adjusting the most cost-effective tuning knob.

Research Agenda

- develop scalable OLTP systems and OLAP-style data management services.
- meta-data manager
- mail server
- testbed for RISC-style components
- work out the APIs for the most important components
- hold a competition for components
- identify the universal glue precisely

Conclusion

- Architectural simplification is overdue and critically needed.
- The true test will be if it can be used broadly in many contexts.
- The root goal is to improve the “gain/pain ratio” of database technology. Tolerate a moderate decrease in gain in return for an orders of magnitude reduction of pain.