Simultaneous Pipelining in QPipe: Exploiting Work Sharing Opportunities Across Queries

Kun Gao
Carnegie Mellon University
kgao@cs.cmu.edu

Stavros Harizopoulos†
MIT CSAIL
stavros@csail.mit.edu

Ippokratis Pandis
Carnegie Mellon University
ipandis@cs.cmu.edu

Vladislav Shkapenyuk†
Rutgers University
vshkap@cs.rutgers.edu

Anastassia Ailamaki
Carnegie Mellon University
natassa@cs.cmu.edu

1. Introduction

Data warehousing and scientific database applications operate on massive datasets and are characterized by complex queries accessing large portions of the database. Concurrent queries often exhibit high data and computation overlap, e.g., they access the same relations on disk, compute similar aggregates, or share intermediate results. Unfortunately, run-time sharing in modern database engines is limited by the paradigm of invoking an independent set of operator instances per query, potentially missing sharing opportunities if the buffer pool evicts data early.

QPipe is a new, operator-centric, relational query engine that can detect and exploit overlap across concurrent queries, at run time [1]. In QPipe, each relational operator is promoted to an independent *micro*-engine (μEngine) with its own resource management and runtime support. Incoming queries break up into as many tasks (or *query packets*) as the nodes of the query tree plan, and queue up in front of each μEngine. Since query packets are self-contained requests, QPipe allows external applications to submit custom packets, bypassing parsing and optimizing phases. Under regular query execution, μEngines work independently and evaluate each query in parallel. Data flow between μEngines occurs through dedicated tuple buffers. μEngines continuously monitor their queue to detect data and work sharing opportunities across queries. Once such an opportunity is detected, only one query packet remains active, performing the overlapping operation, while the results are *simultaneously pipelined* to all consuming queries. More details about sharing opportunities and simultaneous pipelining can be found elsewhere [1].

This system demonstration exposes the key novel features of QPipe, and also provides an intuitive way of visualizing query execution inside the database engine. Unlike modern commercial engines which are typically demonstrated as "black boxes," due to tight integration of system components, QPipe naturally divides query execution into stages, allowing for a visually appealing demonstration. We organize the demo storyline into three parts:

- **Resource utilization and query progress.** For a single query, we show which μEngines are working in parallel, the progress of the query, and the tuple flow between relational operators. For multiple concurrent queries, we show queue status and thread assignment at each μEngine, along with opportunities for sharing data or work across queries.
- **Simultaneously pipelined query execution.** By switching QPipe to *simultaneous pipelining* mode, we demonstrate the entire procedure of sharing overlapping table scans or intermediate result computation across different queries. All possible run-time actions of QPipe are demonstrated through pre-designed scenarios.
- **Ad-hoc query pipelines.** QPipe allows external applications to submit ad-hoc query packets. The user is able to construct and submit custom packets to QPipe through a graphical user interface, and observe how these are evaluated.

2. Demonstration features

We demonstrate QPipe through a graphical user interface that displays dynamically the state of all queries and μEngines in the system. The GUI is implemented in Java Swing. Due to space constraints we do not include screenshots for configuring QPipe or selecting input queries. The user can select a suite of standard TPC-H queries or submit custom queries. QPipe has several parameters accessible through the graphical interface. Each tuple buffer's size can be tuned manually. The μEngines are individually configurable. Users can specify the number of threads as well as the scheduling policy for the μEngines. Once users configure QPipe (or accept the default parameters), they can select which query (or queries) to submit to the engine, or select from a number of pre-specified scenarios (queries along with submission times). Next, we describe the three demo parts in more detail.

Resource utilization and query progress. Figure 1 shows the main execution window when two queries (shown on the upper right corner) are running (the boxes containing μEngines are dragged close together for better

†*Work done while the authors were at Carnegie Mellon University.*

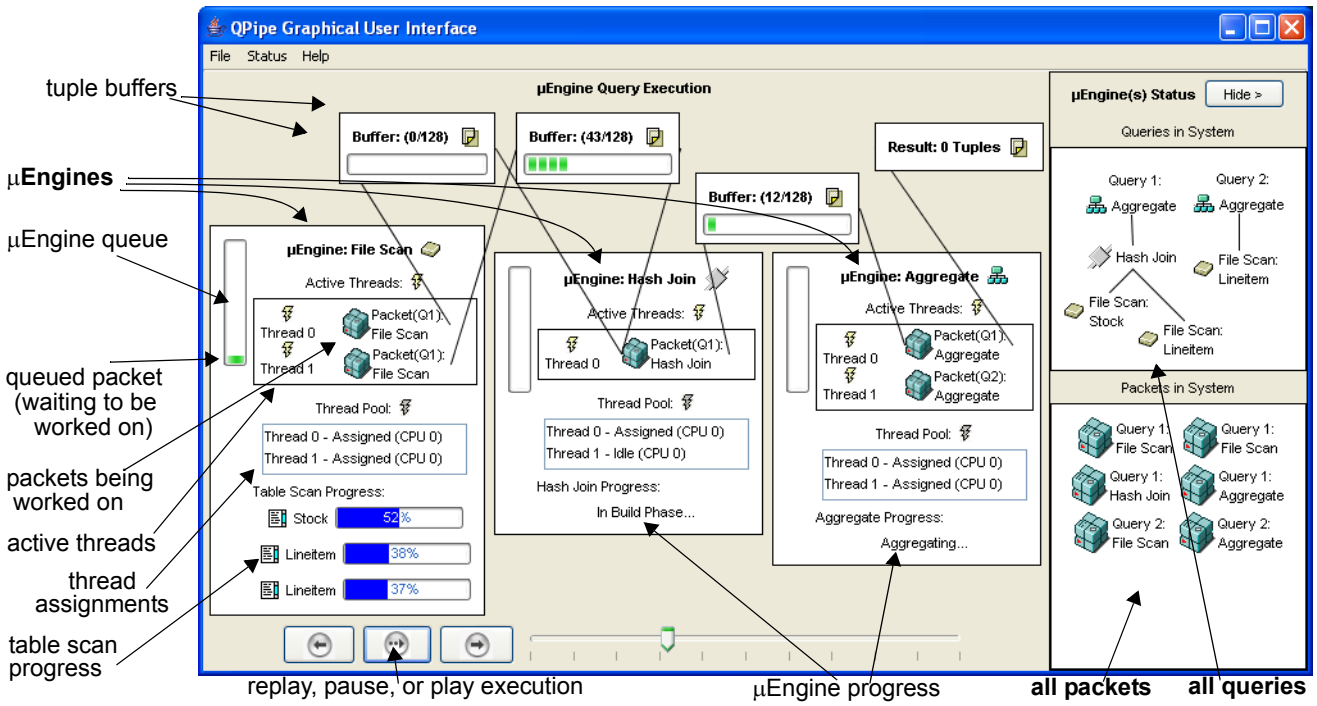

Figure 1. Main execution window (only three µEngines and two queries are shown for simplicity). Clicking on a packet shows the input/output tuple buffers and their state (including final results). Buffers are linked according to the query plan. Both buffers and µEngines can be dragged for improving display.

printing). The execution window shows which threads are working on which packet at each µEngine. For the Scan µEngine, the progress of each table scan is continuously shown. A slider bar allows pausing, playing, and step-by-step replaying of the execution sequence. The top of Figure 1 shows the state of the query intermediate tuple buffers. These are shown by clicking on the individual packets, and can be moved around for better visibility. While on normal operation these buffers fill up/drain at a high rate, by replaying the execution sequence the audience can track in detail the progress of the query in each relational operator. The audience can also track the state of the queues and the thread assignment at each µEngine.

Simultaneously pipelined query execution. Once two or more queries overlap by scanning the same table or by computing the same intermediate result, we point out the opportunity for sharing by raising a flag. The user can verify these opportunities by examining the set of files that are being scanned (for the scan µEngine) or by examining the work other µEngines perform on the packets. By switching QPipe to simultaneous pipelining mode, we demonstrate the entire procedure of sharing overlapping table scans or intermediate result computation across different queries.

Sharing of concurrent scans is advertised in a number of commercial systems (SQL Server 2000, Redbrick, and Teradata), but the ability to share a scan is restricted to unordered scans only (when the order reading the file does not matter). QPipe, however, can also take advantage of concurrent order-sensitive scans. We demonstrate such a scenario, along with scenarios where queries share intermediate computations, such as a join or a sort. When two or more queries share a scan or computation, only one packet performs the work, and the output tuples are simultaneously pipelined to all participating queries (the audience is able to see how the redundant intermediate buffers are removed, and a single packet is linked to multiple consuming queries). More details about scenarios where queries share order-sensitive scans or intermediate result computation can be found elsewhere [1].

Ad-hoc query pipelines. In the last part of the demo, we show how external applications can submit custom query packets to QPipe and have them evaluated. Ad-hoc query packets allow users to take advantage of the functionality of an individual µEngine, without needing to pass through all the modules that are bundled into a traditional DBMS system (such as the parser or the optimizer). We provide an interface to construct arbitrary packets that can be submitted to a single or multiple µEngines from outside the DB engine. We demonstrate an example where the user constructs and submits packets performing filter operations on a set of given files stored in the database.

References

[1] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. "QPipe: A Simultaneously Pipelined Relational Query Engine." In *Proc. SIGMOD,* 2005.