Mortar: Towards Million-Node Data Stream Management

Dionysios Logothetis and Kenneth Yocum
University of California, San Diego

1 Introduction

Compute federations represent a global pool of well-provisioned sensors, continuously emitting system and application-specific data streams, that can be queried for distributed debugging, application control, anomaly detection, measurement and data sampling. Such queries may involve thousands of stream sources distributed across the wide area, implying that node and network failures will be common. This has recently been referred to as the "Internet-Scale Sensing" problem [1].

This poster presents Mortar, a distributed stream processor for building very large, failure resilient queries across thousands of data streams. Mortar's design is driven by the vision of Internet-scale data stream management targeting scale, failure resilience and ease of use. Its primary components are an intuitive, stream-oriented language in which users specify queries, a scalable in-network operator platform, and a failure-resilient data model.

2 Design

Contrary to other large-scale query processors that support only snapshot queries, Mortar is a stream processor; it operates on sliding time or tuple-based windows of values. Users create continuous queries using the Mortar Stream Language (MSL). A user may write an MSL statement to specify the data sources and attributes of interest, the operator type, and the destination of the resulting stream. In MSL, a data source may be a sensor or even the resulting stream of another query, allowing the composition of multiple MSL statements to build complex queries. To support automated actions in system monitoring, MSL also provides *triggers*, user-defined callbacks that execute when specific conditions occur. Figure 1 shows how we can query a Wi-Fi location service for the position of a specific MAC address using only three MSL statements. Select operators filter MAC frames with a specific address locally at the Wi-Fi sensors, while the TopK operator collects the "loudest" frames. The TriLat operator takes these samples and computes a coordinate position based on simple trilateration.

```
E0@local = Select(sniffers.WiFi.frames,
                  T[11]=="00:20:a6:52:bf:17")
                  [tuple=1,slide=1]
E1@root = TopK(E0,E0[3],3) [time=2000,slide=2000]
E2@local = TriLat(E1) [tuple=1,slide=1]
```

Figure 1: The MSL query used for locating a wireless device across the Jigsaw wireless sensor infrastructure.

Mortar builds upon an in-network processing platform that allows the creation of scalable query trees to reduce processing and network load. The runtime compiles an MSL statement to a set of peering operators, across which it establishes physical dataflows. Operators process raw data from sensors and existing streams to produce end results.

Mortar provides best-effort consistency. It does not attempt to salvage data that were in flight during failures. However, its data model allows the user to reason about result quality in terms of completeness and staleness. Mortar annotates tuples with a completeness metric, the percentage of nodes in the system that contributed to a specific window of values. Furthermore, Mortar keeps track of the time that tuples spend in the system and annotates results with an *age*. Its configurable internal stream management policies can balance result staleness and completeness, dynamically adjusting to network conditions. In-network operators buffer intermediate results and emit them based on dynamic timeouts that depend on the observed network latencies.

Mortar tries to provide high-quality answers even in the face of persistent, numerous network or node failures. Its goal is to ensure that operator results capture all constituent data that were *reachable* during the operator's processing window. Unlike other approaches [2, 3, 4], Mortar proposes dynamic tuple striping, a multipath routing scheme that dynamically stripes streams around obstructive network conditions or failures. A physical tree planner creates multiple physical query trees, improving re-

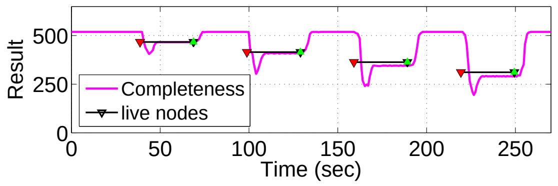

Figure 2: Query performance during rolling failures of 10,20,30, and 40% of 520 nodes.

sult completeness by increasing path diversity. Our experiments show that this technique can improve result completeness by more than 50% compared to existing techniques. Even when 40% of the nodes fail, results include values from almost all live nodes.

Dynamic striping is based on *time-division data partitioning*, a scheme that allows a high-degree of routing freedom. Operators partition data across time, unlike previous approaches that partition data based on its content [3]. In contrast to previous mulitpath-routing approaches, this scheme is independent of the operator type and the data content.

Mortar's failure-resilient data model exists in conjunction with a physical tree planner that, by clustering network coordinates [5], builds network-aware trees. The planning algorithm allows the query root to return the majority of the data quickly, while it reduces the load on the network. A key challenge is to build sibling trees that retain the majority of the clustering of the primary tree while providing node and path diversity. These are competing demands, large changes to the primary will create a less efficient tree. Mortar's planning algorithm does a good job maintaining both properties.

3 Evaluation and future work

Using our prototype we evaluate Mortar's failure resilience on a local-area emulation testbed using queries that source orders of magnitude more nodes than recently published work on existing systems. Figure 2 shows the impact of transient "rolling" failures on result completeness in a network of 520 nodes. In this experiment, we subsequently disconnect and reconnect 10, 20, 30 and 40% of all nodes in the system. Even for 40% failures, results include 88% of the remaining live nodes. Further, evaluating Mortar's physical planner on an Internet-like topology, we observe that the average peer-to-root latency improves by 40% to 50% compared to a random tree.

To validate our stream abstraction, we build a Wi-Fi location service that continuously reports the physical location of mobile devices in a wireless network. We use traces from an existing monitoring infrastructure of 183 sniffers in a large office building. Figure 3 shows the resulting coordinates of the query in Figure 1 plotted on the actual floor map.

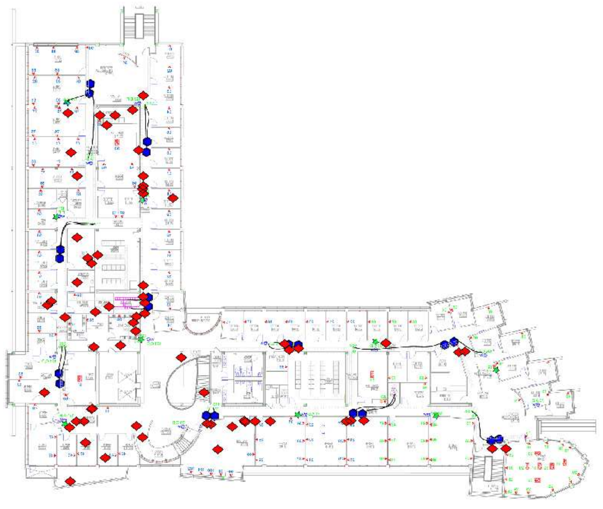

Figure 3: The position of a Wi-Fi user circling the hallways of the UCSD Computer Science department. Diamonds represent query results, octagons Wi-Fi sensors.

Our current Java prototype implements MSL with a variety of operators, dynamic tuple striping, and physical tree planning. Current efforts include a fault-tolerant query installation and recovery mechanism, and the design of stream management policies that incorporate user requirements on result staleness and completeness. Furthermore, Mortar's current time-division data partitioning depends on clock synchronization, an assumption that we seek to eliminate. Finally, we have deployed Mortar on Planetlab, quering slice statistics, while we plan to deploy it on the UCSD Computer Science department Wi-Fi sensor infrastructure.

References

[1] R. N. Murty et al. Towards a dependable architecture for Internet-scale sensing. In *HotDep'06*.

[2] M. Balazinska et al. Fault-tolerance in the Borealis distributed stream processing system. In *SIGMOD'05*.

[3] M. A. Shah et al. Highly available, fault tolerant, parallel dataflows. In *SIGMOD'04*.

[4] S. Madden et al. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI'02*.

[5] F. Dabek et al. Vivaldi: A decentralized network coordinate system. In *SIGCOMM'04*.