EFFICIENT GESTURE STORAGE AND RETRIEVAL FOR MULTIPLE APPLICATIONS USING A RELATIONAL DATA MODEL OF OPEN SOUND CONTROL

Andrew Schmeder

Center for New Music and Audio Technologies (CNMAT)
Department of Music
University of California, Berkeley
1750 Arch Street, Berkeley CA 94720, USA
andy@cnmat.berkeley.edu

ABSTRACT

The role of database technology in real-time interactive music systems is the subject of theoretical elaboration and experimentation in this paper. Example applications are presented along with their informational access requirements. An architecture for a gesture storage/retrieval database is presented using a data-warehousing layer model, featuring a real-time interface that enables operation with interactive systems. An experimental implementation of this design is presented, OSC Stream DB, that uses a relational schema to store and retrieve bundled OSC messages from a PostgreSQL database backend. Its data access layer is optimized for object-retrieval, time jumps/traversals, meta-data based search and content-based search. A case study is conducted with a high-rate touch-sensing gesture interface. A benchmark is created for the real-time interface and tested up to the storage of millions of objects. Recommendations for the use of OSC in real-time gesture applications are given based on the concepts and results of this paper.

1. INTRODUCTION

The use of real-time gesture information in computer music is typically considered as a topic the domain of *interaction design*. In the unified theory of design, *information design* plays an equally important role [10]. Examples within computer music include: the fusion of online music information analysis and generative music creation (as in computer-driven improvisation), dynamic data-driven resonance-model synthesis, and real-time traversal of wave-packet databases (i.e., concatenative synthesis). This paper describes a unified interactive-information architecture for *musical gesture* storage and retrieval.

1.1. The Transistion from Files to Databases

The total quantity of data recorded doubles every three years [8]–increasing in both density and resolution. This leads naturally to the emerging ubiquity of database technologies that enable large quantities of information to be accessed efficiently according to multiple criteria–for example, content-based search and meta-data search. Many components of computer systems previously handled by file systems and file formats are now replaced by databases, including: system configuration, email folders, web sites, and even file systems and file formats themselves (as in XML databases). In light of this development the role of “file” is being redefined as “content format”–a means to exchange data between databases that is information-preserving and implementation-neutral. This suggests that the development of gesture storage/retrieval systems should focus on leveraging database technology rather than attempting to optimize informational access patterns in the file layout. The present work demonstrates that careful attention to efficient design of the data model can make gesture storage/retrieval databases a practical reality.

2. APPLICATIONS OVERVIEW AND THEIR INFORMATIONAL ACCESS REQUIREMENTS

A collection of applications, encompassing: storage, retrieval and search operations with large databases of high-rate interactive gesture information, is summarized here. They can be organized in order of increasing complexity with respect to informational access requirements. For a concise overview see Table 1 (top of next page).

2.1. Gesture Recording

The simplest use-case of a gesture-storage system, recording can be implemented with an append operation on a file. Write latency is typically of low importance.

Informational Access Layer		Data Access Layer				Operational Data Layer
Application	**Actions**	**Granularity**	**Reactivity**	**Index**	**Operation**	**Efficient Data Structure**
Recording	Start/Stop	Incremental	Real-time	Time	Write	Byte Array
Playback	Start/Stop/Rate	Incremental	Real-time	Time	Read	Linked List
Archive/Interchange	Import/Export	Bulk	Offline	Time	Read/Write	Table
Verification	Compare	Incremental	Offline/Real-time	Identifier/Time	Read	B+ Tree
Editing	Modify	Incremental	Offline	Identifier	Read/Write	B+ Tree
Sampling/Remixing	Select/Goto/Rate	Incremental	Real-time	Time/Address	Read	B+/RD Tree
Data Mining/Analysis	Search/Annotate	Bulk/Incremental	Offline	Time/Address/Data	Read/Write	B+/RD/R Tree

Table 1. Applications in order of increasing complexity and their properties organized by layer.

2.2. Gesture Playback

Playback of gestures enables running of interactive programs in the absence of a performer. Its access pattern is the sequential read. A linked list structure enables efficient variable-rate playback including time-reversal (i.e. scrubbing) but not time-jumps. Playback also requires a temporal controller, such as a sample clock (for locked sampling rates) or a "forward synchronizing" real-time scheduler (for random sampling) [9].

2.3. Interchange and Archiving

Interchange is concerned with communicating content between databases or other entities. This topic has received considerable attention [5] and several gesture interchange formats exist including OSC, GDIF, PML and GMS [6]. Ultimately interchange requires standardization of formats, as well as standardization of measurements and representation (i.e., *what* to record [7]). Data access for interchange is bulk, not real-time, and can be uniquely optimized. Database systems often provide a special interface for large import/export operations.

2.4. Systems Verification

Capture and analysis of messaging events can be used to implement unit-testing in real-time environments [2]. This is useful in correctness verification of systems under development, in emulation, or in varying runtime environments. Verification may include a variety of comparisons including between individual input/output messages, groups of messages, and message-response timing behavior.

2.5. Performance Editing

Performance editing is a standard practice in mastering where recordings are "fixed" for problems such performer mistakes or background noise. Typically this is done by editing audio files directly. However, gesture data is a simplified representation of the resulting audio and therefore easier to edit. Editors use search and visualization tools to locate points of interest, and in-place modify operations to change the data. A balanced tree data structure (B+ Tree) enables efficient in-place modify and mid-stream insert operations.

2.6. Gesture Sampling/Remixing

Sampling/remixing is a method for repurposing of digital content. Sampling combines arbitrary jumps to points-in-time and variable-rate playback. The B+ Tree data structure combines a tree (for fast jumps) with a linked-list (for fast sequential traversal). Sampling may also include selection of particular sub-streams within the global gesture-stream (this type of query is demonstrated in the case study of Section 5). Sub-stream selection operators, a type of meta-data-based search, are efficiently provided by an RD Tree, a type of index over sets, for example on collections of semantic tags.

2.7. Data Mining/Analysis

Data mining or analysis tools typically operate in non-real-time and on large chunks of data in matrix formats. Analysis can include writing new data back into a stream as an annotation layer. Examples include segmentation, clustering, and data modeling. The data access includes content-based search operations. In particular, queries over multidimensional numeric data are efficiently supported by the R Tree, a data structure for searching and traversal of spatial data.

3. THE OSC STREAM DB ARCHITECTURE

An architecture for a gesture storage/retrieval system is given in Figure 1. The system diagram is separated into layers using the operational/data-access/informational-access terminology from data warehousing theory [4]. The contribution of this work is the central data access layer that makes the database backend (provided by PostgreSQL [1] and the GiST index [3]) useful for applications. This implementation is referred to as OSC Stream DB. OSC is Open Sound Control [14], an open content format for real-time gesture data. OSC features high temporal accuracy using message timetags, and has a simple labeling method that maps controller components to path-style strings.

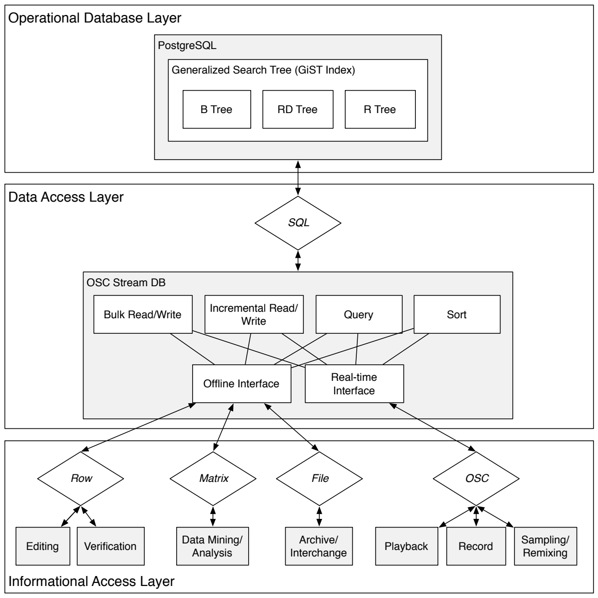

Figure 1. Architecture of OSC Stream DB

3.1. A Relational Model of OSC Streams

We define the streams to generalize over unidirectional, bidirectional and multicast transports where each message has a source and target endpoint identifier. Each framed packet in the stream is a timestamped OSC bundle containing one or more messages. (Figure 2). While the OSC Specification [14] permits messages to exist outside of the bundle context, non-bundled messages are not supported by OSC Stream DB.

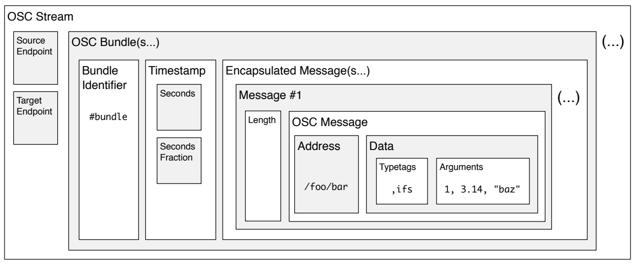

Figure 2. Structure of an OSC Bundle and Message

Translating the OSC object-model of Figure 2 into normalized relational form yields the schema in Figure 3. The schema uses the following strategies to ensure efficient data handling:

- Separation of address and typetag strings into separate tables: the majority of bit-sparsity in OSC packets is attributed to these fields. In this design long address impose no additional cost.
- Message data is stored exactly as-received in the "data_binary" field, enabling exact recall of entire OSC bundles within a single, multi-way joined table select.

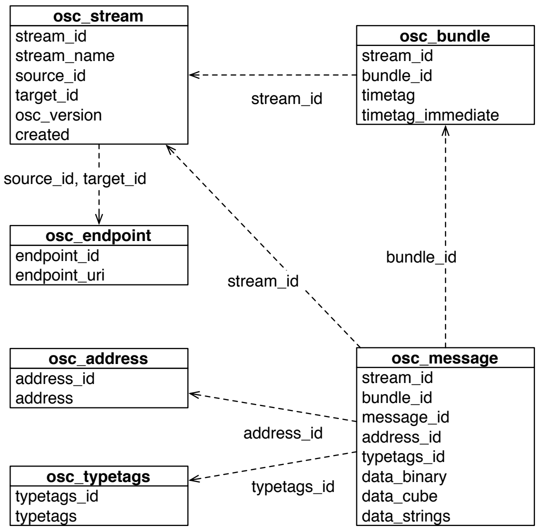

Figure 3. Normalized schema for OSC Streams

3.2. Interface Design

OSC Stream DB features two interfaces: one real-time for hybrid informational+interactive designs, and one offline for data mining/analysis.

3.3. Offline/Bulk Interface

The offline interface is implemented with language-native vector/matrix formats and database system client libraries as the data access bridge. Its implementation is straightforward in any modern programming language (e.g., by using JDBC from Matlab).

For purposes of file interchange, OSC Stream DB can read from or write to a binary file using framed OSC bundles as the content format.

3.4. Real-Time Interface

The real-time interface to OSC Stream DB is accessed interactively over network messaging of OSC-encoded packets. It contains a collection of ports, dedicated by function: for commands such as search queries, for writing information to the database, and for receiving results of read operations. A schematic depicting the flow of information between these ports and a client application is given in Figure 4.

To record output from a gesture source, the client streams bundles into the write-port. To read back the gesture data, the client sends messages to the command-port to specify the query parameters: for example, by placing and seeking with a time-cursor, configuring of search filters, and requesting data in small blocks of data to be streamed back on the read-port. The OSC address space for this set of command operations is given in Table 2.

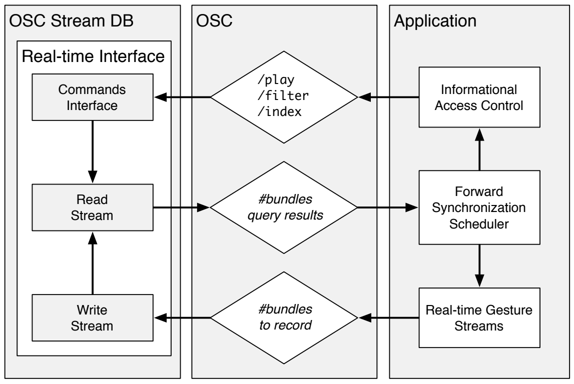

Figure 4. Real-time interface ports and components

Message	Typetag:Argument(s...)
`/read`	`tt:min,max`
`/play`	`tttf:min,max,ref,rate`
`/seek/time`	`t:time`
`/seek/id`	`i:bundle-id`
`/seek/(start|end)`	-
`/seek/(min|max)`	-
`/seek/(prev|next)`	`i:step-by`
`/filter/address`	`s:pattern ...`
`/filter/numbers`	`[f][f]:bounding-box`
`/filter/strings`	`s:pattern ...`
`/filter/...`	`...`

Table 2. Command interface to OSC Stream DB.

3.4.1. *Real-time playback*

Real-time playback requires a temporal controller, and this must reside in the application (on the client side). By applying a map from recorded OSC bundle timetags to real-time future timetags, OSC Stream DB enables the eventual reconstruction of temporal information (even to the point of "sample accuracy") after processing by the client-side forward-synchronizing scheduler.

In the case of variable-rate playback, the timetag transformation is an instantaneous translation and scaling of time. Algorithm 1 shows in pseudo-code how this can be accomplished with piecewise-linear adjustment of the rate.

As seen in the benchmarks of Section 6, a setting of d_f to 10 milliseconds is sufficient to account for worst-case roundtrip latency. d_t is adjustable anywhere from 10-100 milliseconds depending on desired response time.

3.5. Stream Data Models: SDIF vs OSC

Given the recent efforts to adapt SDIF [13] to gesture storage/retrieval (i.e., GDIF [7]), it is worth noting the difference between the SDIF and OSC data models at this point (see Table 3). In spite of differences in stream organization, the basic data models are essentially equivalent. However, the formats are not: SDIF was designed around operational concerns related to efficient playback. The format of OSC is designed to be a self-contained structure for stateless processing, making it a better fit for the modern conception of files as content-format. OSC also has significantly more complex meta-data that enables interesting queries over substreams.

```
Input: $d_t$ tick interval, $d_f$ forward sync delay
Output: bundles $b$
1  $q$ = Select $min(b.timetag)$;
2  while running do
3      $t \leftarrow now()$;
4      $r \leftarrow r_t$;
5      Select $b$ where $q \leq b.timetag < (q + d_t \times r)$;
6      foreach $b$ do
7          $b_i.timetag \leftarrow ((b_i.timetag - q)/r) + (t + d_f)$;
8          Select $m$ where $m.bundle\_id = b.bundle\_id$;
9          foreach $m$ do
10             Encapsulate $m_j$ in $b_i$;
11         end
12         Schedule $b_i$ at $b_i.timetag$;
13     end
14     $q \leftarrow q + d_t \times r$;
15     Wait $d_t$;
16 end
```

Algorithm 1. Realtime streaming playback with piecewise-linear variable rate control. Note that mathematical operations on OSC timetags require special handling to ensure sufficient numerical precision is maintained throughout.

Design	Property	SDIF	OSC
Data Model	Streams	Many	One
	Stream Type	Homogenous	Heterogeneous
	Atomic Unit	Frame	Bundle
	Datum Unit	Matrix	Message
	Timestamp	Relative Time	Absolute Time
	Data Type	Matrix Type	Typetags
Data Format	File Format	SDIF File	SLIP+OSC
	File Structure	Linked List	Byte Array
	Implementation	Complex	Trivial
Meta-data	Field	Matrix Type	Address
	Components	Single	Many
	Regulation	Predefined	Open
	Organization	Flat	Hierarchical

Table 3. Comparison of SDIF and OSC formats.

4. EFFICIENT DATA ACCESS, QUERIES AND SORTING

The cube metaphor (Figure 5) is a spatial conceptualization of how a relational database combines indexes over multiple attributes. Queries on the cube are spatial selection operators, e.g., a point, a slice plane, or a sub-cube volume.

In OSC Stream DB, the primary attributes of interest are time, address and data. Additionally, note there are also more attributes of use including: stream identifier, source/target identifier, bundle identifier, message identifier, and typetags.

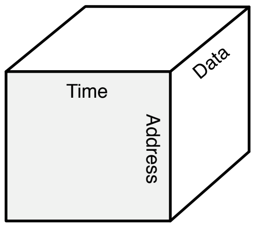

Figure 5. Cube metaphor of dimensional queries, sorting

4.1. Time Index Access

Time is represented by a linear one-dimensional numeric attribute. Queries over time are efficiently handled by the B+ Tree index, providing $O(log(N))$ random access and $O(1)$ sequential access. Types of time queries include:

- **Point-in-time**: Locate nearest bundle to a time-value (Figure 6 and `/seek/time`).
- **Sequential traversal**: Move a "cursor" forward or backward in time (`/seek/(prev|next)`).
- **Extreme values**: Find first, last item in sequence (`/seek/(min|max)`).
- **Interval Selection**: Select and order all bundles occurring between two points in time (`/read`, `/play`).

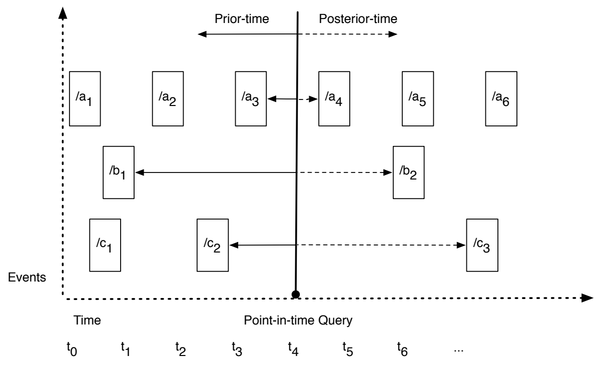

Figure 6. Point-in-time query locating nearest events events

4.2. Address Pattern Matching

Addresses in OSC are multi-component paths. Queries over the address are a type of meta-data search. The OSC Specification [14] describes a pattern-matching syntax using wildcard operators: *, which matches a single address component or partial component, and //, a path-traversal operator that matches any number of complete address components. These operators provide selection capability that is both vertically and horizontally oriented with respect to the the path hierarchy (Figure 7). A data structure that supports these types of queries in $O(log(N))$ time is the *label tree*. The label tree in PostgreSQL (see `contrib/ltree` in [1]) is implemented using the RD Tree, a type of index over sets (i.e., collections of strings).

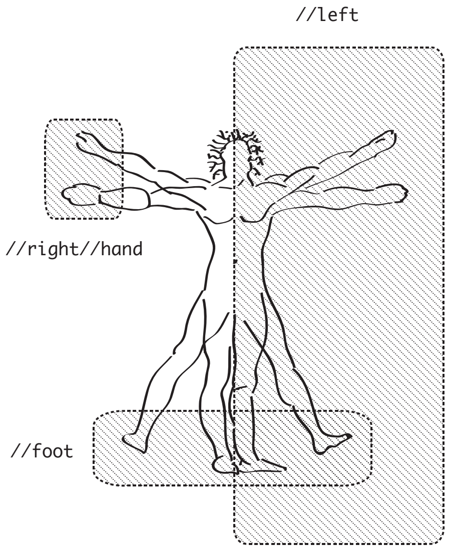

Figure 7. Horizontal and vertical selections on a label tree

4.3. Data Content Search, Spatial Selectors

Queries on the message data section (sometimes called "arguments") are a *content-based* search. In the case of OSC this can mean searching for packets with numeric data in a specified range, or searching for text content matching a pattern.

4.3.1. Spatial queries on numeric data

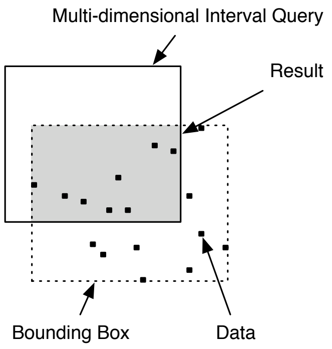

Figure 8. Multidimensional numeric data selection with the n-cube-contains operator

All number classes are first encoded into double-precision floating point, which is a superclass of all formats that OSC supports. Booleans and nulls are included with the map $\{T \rightarrow 1, F \rightarrow 0, N \rightarrow -1\}$. The resulting array of numbers, which can be any length, is stored as an zero-volume hypercube or *ncube* (see `contrib/cube` in [1]).

The ncube data type is efficiently queried by the R Tree, a type of index for spatial data. The fundamental selection operator for a spatial query is *contained-in*, which tests for intersection of data with a bounding ncube (see Figure 8). The ncube data type also supports binary operations (intersect, union) between ncubes of different dimensions.

5. CASE STUDY

This case study is designed to demonstrate how OSC Stream DB can be used to record, query and playback gesture data. The study includes the design of an OSC address space for a controller that annotates the stream with event markers enabling event-segments to be efficiently located and extracted.

5.1. The SLAB32 Interface

The SLAB32 is a touchpad array designed and performed by David Wessel [12] (Figure 9, top left). The SLAB uses a digital audio connection to stream pressure and position information from each pad to a host computer. On the host, this stream was sampled at the rate of 1000 Hz into discrete events, timestamped in OSC bundles.

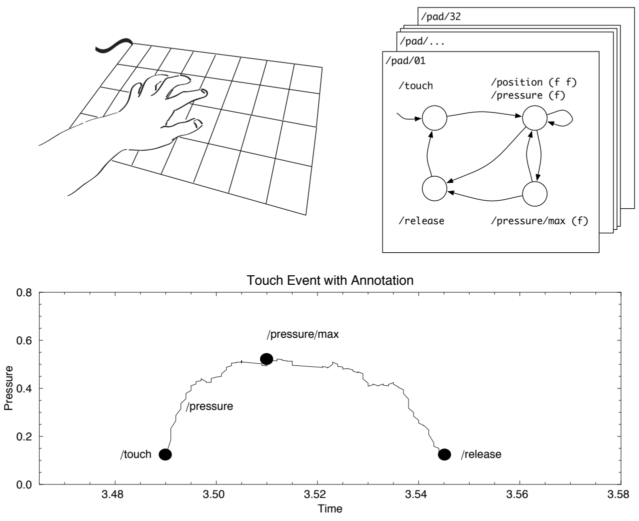

Figure 9. Hand activates 1-5 touchpads, transmitted in digital audio. Finite state machine of a touch event. Annotated plot of a touch event.

5.2. Segmenting Touch Events

A finite-state machine representing a single touch-event was designed (Figure 9, top right). Each node is labeled with the corresponding OSC message generated by that state. The `/touch` and `/release` states can be used as segmentation markers to perform a query on the gesture data. For example to find and extract a particular gesture (Figure 9 bottom), the following commands are used:

```
/filter/address "/touch"
q1 ← /seek/prev
/filter/address "/release"
q2 ← /seek/next
/filter/address "/pressure"
/read (q1, q2)
```

5.3. Recording an Annotation Layer

The `/pressure/max` state is different from other states because it is obtained from a moving-average filter that introduces causal delay. A unique advantage of the B+ Tree is that it enables OSC Stream DB to store events in non-sequential time order. Thus, when a pressure maximum is detected, it is recorded to the database using a timetag that is adjusted to compensate for the causal delay (i.e., by subtracting a duration equal to half the window size).

6. PERFORMANCE EVALUATION

The practicality of a real-time gesture storage/retrieval system depends on its ability to complete data-access operations, both sequential and random lookups, in faster-than real-time and with low delay. Also, this performance needs to be retained as the size of the database increases. The basic design of this benchmark test is to measure stream throughput and worst-case random-read latency over time, during which the database is rapidly being filled with millions of records.

6.1. The Benchmark Test Stream

The test stream is a synthetic stream that has a nominal 1000 Hz bundle rate, i.e., the bundles are timestamped in sequential intervals of 1 millisecond. Each bundle contains ten messages with a data-payload of randomly generated floating-point values, (`/test/(1-10) f f f`). When this stream is running at the nominal rate, it is approximately the same bandwidth as in the SLAB32 case-study (Section 5) when all ten-fingers are on the device. The test stream properties are summarized in Table 4.

Property	Value
Bundle rate	1K per Second
Message rate	10K per Second
Bundle size	10 Messages * 34 Bytes = 340 Bytes
Data rate	2.7M bits per Second

Table 4. Benchmark stream at 1000 Hz nominal

6.2. Write Throughput Benchmark

Write throughput is measured by streaming bundles into the write-port at the maximum possible rate. OSC Stream DB optimizes inserts by collecting rows for ten seconds before executing the commit. The overhead of updating schema indexes makes this operation both computationally and bandwidth constrained.

6.3. Read Throughput Benchmark

The read benchmark uses randomly distributed start-times taken anywhere in the stream, and selects the following bundles corresponding to d_t = 100 milliseconds (100 bundles spanning 0.1 seconds of at the nominal rate). The read test includes complete reconstruction of the original OSC bundles and streaming of the results back over the read-port. Read throughput is primarily bandwidth constrained.

6.4. Realtime Delay Benchmark

Round-trip response time from a random query to the return of its first selected row is measured while the system is simultaneously undergoing the read throughput test. The upper bound of this delay is computed with repeated measurements to obtain a high probability bound on the worst-case latency.

6.5. Conditions

The computer used in this test is an Apple Mac Pro 8-core Desktop. (Dual Quad-core Intel Xeon “Nehalem” 3GHz, 8G RAM, single SATA disk). OSC messages were transmitted over the loopback network interface. PostgreSQL 8.3 was configured using the pgtune script with the data warehouse (“DW”) setting [11]. The OSX kernel was configured to enable large shared memory segments. The benchmark application was implemented in MaxMSP. The OSC Stream DB process is a highly optimized multi-threaded program written in C using all-binary interfaces for OSC networking and the PostgreSQL backend.

6.5.1. Optimization for parallel processing

The benchmark program was configured to support parallel operations by enabling multiple instances of the OSC Stream DB process to run simultaneously. Task-parallelism was achieved with round-robin distribution. Reformulating the process using data-parallelism with the map-reduce pattern gave no significant change in throughput–however, such a strategy should scale well on a system (or cluster) with multiple disk drives. In write performance, there was a clear advantage to multiple instances, although it was found to diminish for $N > 2$ and have no improvement after $N > 5$. Read performance was fastest for only $N = 1$; allowing parallel sequential reads did not improve throughput nor latency (Figure 10).

6.6. Performance Results

The benchmark was run continuously under the optimized conditions such that the total size of the database was within the effective cache capability of the machine. At the peak size the database contained approximately 1.4 million bundles and 14 million individual messages, having an on-disk size of about 4Gb. The performance results are plotted in Figure 11.

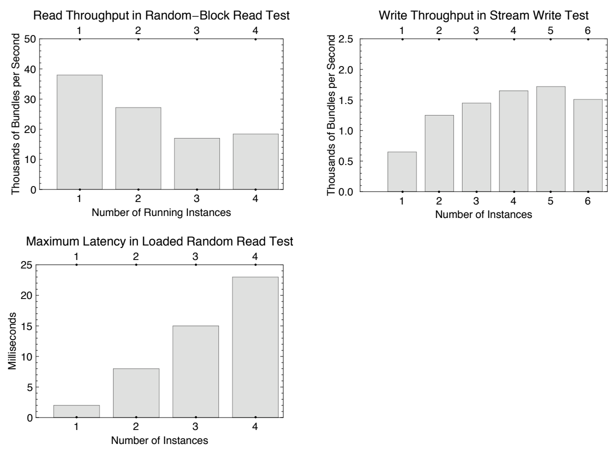

Figure 10. Performance in parallel mode

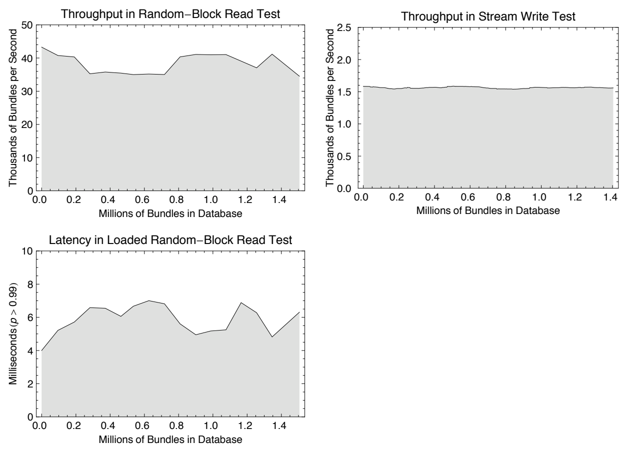

Figure 11. Performance under increasing database size

The random-block read test shows that the system is capable of resynthesizing the gesture stream as fast as 40x over real-time with respect to the test-stream. Additionally, the round-trip latency of random seeks is well within the range desired for interactive control. Write performance is a heavy operation, but still capable of exceeding the nominal rate by about 50%.

7. CONCLUSION

The OSC Stream DB system demonstrates that gesture storage/retrieval systems can be implemented for real-time systems with sufficient performance and capacity by leveraging commonly available relational database systems. The design of this architecture elucidates the types of informational access patterns and supporting components that bridge between informational systems and interactive systems. This development advances the possibilities for applications that leverage informational repositories in conjunction with real-time interactivity.

7.1. Recommendations for use of OSC in Joint Information/Interaction Designs

The utility of gesture storage-retreival systems can be maximized by appropriate use of the OSC format in the design of new address spaces. The following recommendations are put forth:

- Always use bundles. Non-bundled messages have no temporal information.
- Use real timestamps everywhere and synchronize clocks. Do not use "immediate" timestamps. Physical real-time ensures temporal information can be integrated in heterogeneous and randomly-sampled environments.
- Design address spaces according to informational access requirements: queries over address patterns are the most powerful and efficient means to locate data of interest. Use annotation layers to mark points of interest in the gesture stream.

7.2. Caveats and Limitations

In spite of enthusiasm for the OSC Stream DB implementation, from a practical standpoint its use is hindered by many factors, including: expertise required to correctly install and optimize a relational database system, and the lack of transparent end-to-end handling of temporal information in platforms used to build client applications. We should encourage discussion and awareness of the needs of end-to-end temporal semantics in all types of software systems including: operating systems, web services, relational databases and application-building platforms.

7.3. Future Work

Perhaps the most significant remaining open area of inquiry concerns alternatives and enhancements to the basic relational model. For example, by giving a description of SDIF frames in OSC (using the proposed OSC 1.1 matrix type), source material from either format could be merged. Ultimately, divergence from the OSC-based schema may prove necessary. In particular, OSC address pattern matching, while reasonably expressive, is not capable of representing the full range of possible semantic queries over a subject-predicate-object ontology.

8. ACKNOWLEDGEMENTS

Adrian Freed for ideas and motivation, the anonymous reviewers for useful suggestions, the and UC Berkeley Parallel Computing Laboratory for the test machine.

9. REFERENCES

[1] "PostgreSQL," http://www.postgresql.org/.

[2] A. Chaudhary, "Automated testing of open-source music software with open sound world and open sound control," in *Proceedings of the ICMC*, 2005.

[3] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, *Generalized Search Trees for Database Systems*. The MIT Press, 2005.

[4] W. H. Inmon, *Building the Data Warehouse*, 4th ed. Wiley Publishing Inc., 2005.

[5] A. Jensenius, A. Camurri, N. Castagné, E. Maestre, J. Malloch, D. McGilvray, D. Schwarz, and M. Wright, "Panel: The need of formats for streaming and storing music-related movement and gesture data," in *Proceedings of the ICMC*, 2007.

[6] A. Jensenius, N. Castagné, A. Camurri, E. Maestre, J. Malloch, and D. McGilvray, "A summary of formats for streaming and storing music-related movement and gesture data," in *Enactive Conference*, 2007.

[7] A. R. Jensenius, K. Nymoen, and R. I. Godøy, "A multilayered gdif-based setup for studying coarticulation in the movements of musicians," in *Proceedings of the ICMC*, 2008.

[8] P. Lyman and H. R. Varian, "How much information? 2003," http://www.sims.berkeley.edu/research/projects/how-much-info-2003/, Oct 2003.

[9] A. Schmeder and A. Freed, "Implementation and Applications of Open Sound Control Timestamps," in *Proceedings of the ICMC*. Belfast, UK: ICMA, 2008, pp. 655–658.

[10] N. Shedroff, *Information Interaction Design: A Unified Field Theory of Design*. MIT Press, 1999.

[11] G. Smith, "Pgtune," http://pgfoundry.org/projects/pgtune/.

[12] D. Wessel, R. Avizienis, A. Freed, and M. Wright, "A force sensitive multi-touch array supporting multiple 2-d musical control structures," in *NIME*, 2007.

[13] M. Wright, A. Chaudhary, A. Freed, S. Khoury, and D. Wessel., "Audio applications of the sound description interchange format standard," in *Audio Engineering Society 107th Convention, New York*, 1999.

[14] M. Wright, "Open Sound Control 1.0 Specification," http://opensoundcontrol.org/spec-1_0, 2002.