Approximation and Interaction: A Progressive’s View

JOE HELLERSTEIN
Outline

1. Perspective
2. Interactive as Distributed
3. Async Interaction
4. CALM Progress
5. More Progress
Perspective

- Distributed Systems
- Visualization/Interaction
- Machine Learning

Through a data-centric, declarative lens
Systems and Services

- Many discrete low-latency tasks $x \rightarrow T(x)$
- Multi-user
- Concurrent, session-oriented
- Mutable state
Services as Stream Queries

- One stream \((uid, sid, x) \rightarrow Q(uid, sid, x)\)
- Partitioned by user, session
- State evolution as a log
  - the “kappa architecture”?
- Goes deeper:
  - Both system internals & application logic implemented as stream queries

[ACHM11, CMA+12]
**Progressive Systems: Monotonic by Nature**

- Most services make forward progress only:
  - *monotonic* queries over unbounded streams
  - New inputs only cause new outputs – no retractions!

- Benefits: replication, partitioning, lineage debugging…
  - Declarative networking, database & distributed systems
    - [P2 LCH+05], [DSN CPT+07], [Evita CCHM08], [BOOM ACC+10a], [IDo ACC+10b],
    - [ExSpan ZST+10], [LogicBlox AtCG+15]
  - Convergent Replicated Data Types
    - [Treedoc LPS10], [CRDT SPBZ11], [RedBlue LPC+12]
  - CALM Theorem: Coordination-Free Consistency
    - [Hel10], [ANVdB13], [ZGL12], [AKNZ16]
Progressive Systems

- How might this be relevant to long-running interactive tasks?

- Surprise (?) : that’s where it all started!
A Progressive’s Progress

- Online Aggregation
- Adaptive Dataflow
- Stream Processing
- Declarative Networking
  Declarative Distributed Systems
A Progressive’s Progress

How does the later work on declarativity and monotonicity reflect back? On Interaction? Approximation?

Results and open questions…
Outline

1. Perspective
2. Interactive as Distributed
3. Async Interaction
4. Outline Item
5. Outline Item
Interfaces c. 1995 ... and in our era of Big Data

- Lack of feedback

- Coarse-grained user control
  - query
  - cancel

- Online Aggregation can help
  - Continuous approximation
  - But what is the User Experience?
An Interface for Online Aggregation

- Progressive animation
  - approximation
  - confidence
  - rate of change
- Visual update-in-place
  - mutable state!

With thanks to Bruce Lo, 1997
Interaction Starts with Eye

- Output is progressively interpreted by a human
- Human input is also an important stream
- What is in the middle of this control loop?
The Model Human Processor

Card, Moran, Newell ’83 [CMN83]
A Distributed System...
...With Distributed Systems Problems

- Lost messages
- Batched message
- Reordered messages
- Performance variance, component failure
- Heterogeneous storage and compute
Architectural Concerns

- Huge data volumes
- Large-scale computation
- Low BW, Intermittent
- Limited Memory
- High context switch cost
- High latency
Consistency Challenges

Evolving Distributed State

\[ \sum_{i,t} S_{i,t} \]

Evolving distilled visual representation

\[ V_t = f(\sum_{i,t} S_{i,t}) \]

Lossy memory of visual and semantic history

\[ \hat{V}_t \quad \hat{M}_t \]

Lossy perception and semantic understanding

High latency
I’m Living This
<table>
<thead>
<tr>
<th>Stock Categories</th>
<th>Date/Time</th>
<th>Date/Time</th>
<th>Date/Time</th>
<th>0-437</th>
<th>0-2.89k</th>
<th>0-698,779</th>
<th>294,512-294,512</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA</td>
<td>7/6/2015</td>
<td>14:10:58</td>
<td>7/6/2015</td>
<td>7:04</td>
<td>115.479</td>
<td>528</td>
<td>69672.91</td>
</tr>
<tr>
<td>TOEHY</td>
<td>7/1/2015</td>
<td>4:00:23</td>
<td>7/6/2015</td>
<td>8:04</td>
<td>194.672</td>
<td>1190</td>
<td>231659.68</td>
</tr>
<tr>
<td>ADS</td>
<td>7/7/2015</td>
<td>19:28:47</td>
<td>7/2/2015</td>
<td>9:28</td>
<td>89.582</td>
<td>1338</td>
<td>119868.72</td>
</tr>
<tr>
<td>SSP</td>
<td>7/2/2015</td>
<td>8:87:28</td>
<td>7/5/2015</td>
<td>9:19</td>
<td>82.381</td>
<td>854</td>
<td>70353.37</td>
</tr>
<tr>
<td>CAE</td>
<td>7/6/2015</td>
<td>1:22:22</td>
<td>7/3/2015</td>
<td>14:45</td>
<td>211.085</td>
<td>136</td>
<td>28696.68</td>
</tr>
<tr>
<td>NOK</td>
<td>7/3/2015</td>
<td>11:02:9</td>
<td>7/5/2015</td>
<td>2:09</td>
<td>17.114</td>
<td>753</td>
<td>12886.84</td>
</tr>
<tr>
<td>GOOGGL</td>
<td>7/2/2015</td>
<td>14:09:55</td>
<td>7/6/2015</td>
<td>8:04</td>
<td>58.165</td>
<td>1378</td>
<td>80151.37</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>7/6/2015</td>
<td>1:59:08</td>
<td>7/7/2015</td>
<td>9:08</td>
<td>253.315</td>
<td>553</td>
<td>140093.2</td>
</tr>
<tr>
<td>ATRO</td>
<td>7/7/2015</td>
<td>22:07:8</td>
<td>7/4/2015</td>
<td>6:00</td>
<td>183.792</td>
<td>827</td>
<td>151995.98</td>
</tr>
<tr>
<td>TOT</td>
<td>7/4/2015</td>
<td>2:45:57</td>
<td>7/6/2015</td>
<td>13:34</td>
<td>133.25</td>
<td>404</td>
<td>53833</td>
</tr>
<tr>
<td>DCO</td>
<td>7/5/2015</td>
<td>7:11:29</td>
<td>7/4/2015</td>
<td>15:54</td>
<td>117.854</td>
<td>1223</td>
<td>143157.84</td>
</tr>
<tr>
<td>RSR</td>
<td>7/6/2015</td>
<td>23:41:21</td>
<td>7/1/2015</td>
<td>9:00</td>
<td>87.571</td>
<td>246</td>
<td>21542.47</td>
</tr>
</tbody>
</table>
Outline

1. Perspective
2. Interactive as Distributed
3. Async Interaction
4. CALM Progress
5. More Progress
Asynchronous Data Visualization

- **Chronicled Interactions**

- Joint work with Yifan Wu, Larry Xu, Eugene Wu, Remco Chang
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response

![Graph showing stock price over years]

![Diagram illustrating user interaction with a visualization interface]
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response
A Simple (?) Case: High-Latency Interaction

- Attach a visualization interface to a “big data” system
- One option: serial request/response
Alternative: Asynchronous Interaction

- Immediate rendering
  - out-of-order response arrival
  - lower-latency feedback

- Confusing!
  - How, specifically?
User State

1. Buttons I pushed
2. Requests I caused
3. Responses on display
User State

1. Buttons I pushed
2. Requests I caused
3. Responses on display
4. Correspondences between requests and responses
User State

1. Buttons I pushed
2. Requests I caused
3. Responses on display
4. Correspondences between requests and responses

Typical assumption: in the user’s head
User State: the Serial Case

1. Buttons I pushed  (1)
2. Requests I caused  (1)
3. Responses on display  (1)
4. Correspondences between requests and responses

Reasonable assumption: in the user’s head
User State: the Async Case

1. Buttons I pushed (7)
2. Requests I caused (5)
3. Responses on display (3)
4. Correspondences between requests and responses

Unreasonable assumption: in the user’s head

Lossy memory of visual and semantic history

Intermittent/Lossy perception

Latency
User State: the Async Case

1. Buttons I pushed (7)
2. Requests I caused (5)
3. Responses on display (3)
4. Correspondences between requests and responses

<table>
<thead>
<tr>
<th>ButtonID</th>
<th>X_range</th>
<th>Y_range</th>
<th>API</th>
<th>args</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[13,73]</td>
<td>[10,20]</td>
<td>fetch</td>
<td>['March']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API name</th>
<th>timestamp</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch</td>
<td>21</td>
<td>['June']</td>
</tr>
<tr>
<td>fetch</td>
<td>22</td>
<td>['March']</td>
</tr>
<tr>
<td>fetch</td>
<td>23</td>
<td>['May']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>month</th>
<th>call_time</th>
<th>results</th>
</tr>
</thead>
<tbody>
<tr>
<td>'June'</td>
<td>21</td>
<td>[24, 16, ...]</td>
</tr>
<tr>
<td>'May'</td>
<td>23</td>
<td>[14, 22, ...]</td>
</tr>
<tr>
<td>'March'</td>
<td>22</td>
<td>[6, 13, ...]</td>
</tr>
</tbody>
</table>

Lossy memory of visual and semantic history

Intermittent/Lossy perception

Latency

$V_t$, $M_t$
User State: the Async Case

1. Buttons I pushed (7)
2. Requests I caused (5)
3. Responses on display (3)
4. Correspondences between requests and responses

Visualize the async state!

Lossy memory of visual and semantic history

Intermittent/Lossy perception

Latency
Chronicled Interaction: Overlay

- Option 2: overlaid async chronicle
- Immediate rendering
  - out-of-order response arrival
  - lower-latency feedback
- Order-restoring visualization
  - recency => color
  - request/response correspondence: color
  - bounded history
Chronicled Interaction: Small Multiples

- Option 3: spatial async chronicle
- Immediate rendering
  - out-of-order response arrival
  - lower-latency feedback
- Order-restoring visualization
  - recency => color
  - request/response correspondence: label
  - bounded history
User Studies: Completion Time

- High latency (blue):
  - Chronicles improve completion time vs. Serial
- Low latency (red):
  - Serial dominates Chronicles

Figure 12. Each chart visualizes median task completion time with 95% CI (y-axis), for the conditions within an experiment group: designs (x-axis), and latencies (hue). The charts are faceted by task types across the charts.
User Studies: Concurrency x Completion

- With good interfaces, users work concurrently
  - And finish faster
- Bad interfaces cause self-serialization

![Graph showing completion time against concurrency for different tasks](image)

**Figure 13.** A scatter plot of accurately completed tasks’ completion time against concurrency, faceted by tasks, under high latency. Pearson’s $r$: threshold $r = -0.60, p < 0.00001$, maximum $r = -0.60, p < 0.00001$, and trend $r = -0.50, p < 0.00001$. In addition to the negative correlation between completion time and concurrency, the points skew slightly to the upper left for the maximum as compared to the two others, indicating that maximum takes longer and discourages asynchronous interactions.
Design Principles

“Progressive” visualization:

- Interaction history and output history both visualized ("chronicle")
- Monotone evolution of vis tracks the march of time (dark → light → gone)

Program state is data: easy to visualize state, history

- System “internals”: request/response buffers
- Chronicled ordering of events
- Colors allow human processor to replicate the async join

All makes visualization easier to understand

- Analogous to how we think about distributed systems!
Design Patterns: “Building On Quicksand”

- Experiences from Microsoft and Amazon in the late oughts
  - E.g. Amazon Dynamo

[Helland/Campbell 2009]
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Orange</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
The Classical Solution

- Coordination — i.e., global agreement
  - Two-Phase Commit
  - Paxos
  - BSP barriers
- Basically, ensure all nodes agree on separation in time
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Apple 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple 1</td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Apple</td>
<td>1</td>
</tr>
</tbody>
</table>

![Image of stopwatch](image)
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍊</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>1</td>
</tr>
</tbody>
</table>
What’s So Slow ‘Bout Peace Love and Understanding?
What’s So Slow ‘Bout Peace Love and Understanding?
Design Pattern: ACID 2.0

Theme: Translate state mutation into

- Associative
- Commutative
- Idempotent
- Distributed

... logs of application-oriented requests
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>1</td>
</tr>
<tr>
<td>🍊</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>1</td>
</tr>
<tr>
<td>🍊</td>
<td>1</td>
</tr>
</tbody>
</table>
Formalism: The CALM Theorem

Theorem: CALM (Consistency As Logical Monotonicity).

The following are equivalent computational classes:

1. Problems that do not require coordination for distributed consistency
2. Problems expressible in Monotonic Logic

Said differently:

Eventual Consistency Possible iff Problem is Monotone

[Hellerstein PODS ‘09]
[ANV PODS ‘11, JACM ‘13]
[ZGL PODS ‘12]
[AKN PODS14, JACM16]
The Expressive Power of CALM

- Conjecture: Coordination-Free $\equiv$ PTIME
  - Via Immerman/Vardi (semi-positive Datalog with successor $= \text{PTIME}$)

- In a better world, we’d probably never use/need coordination
  - We are slaves to the legacy of Read/Write I/O assumptions
CALM Design Patterns

- Many programs can be written monotonically
  - Monotonic = Coordination-Free = Embarrassingly Parallel.
  - No need for Lamport clocks, 2PC, “time” of any kind

- Logic + Lattices (CRDTs)
  - With lattice homomorphisms and monotone functions  [CMA SOCC12]
So What is Time For?

Tiger
by Max Brand

A Tale of the Manhattan Jungle

The Time Professor
by Ray Cummings

“I do know what time is,” Tubby declared. He paused. “Time,” he added slowly—“time is what keeps everything from happening at once. I know that—I seen it in print, too.”

The first man stared in awe.
Back to the Point

- What should be *progressively* rendered
  - Visualizations you can make order- and batch-insensitive
- What should be separated in time — or space?.
- And why?!
Separation Can Be Good

- We may want to demarcate “sessions” or “tasks”
  - Really just a “partitioning key”, not ordering.

- We may want to record a sequence
  - Again, may simply be annotation data for human consumption

- That’s OK! Humans exist in space and time
  - Even if most tasks are embarassingly parallel
Layout in Time and Space

- Either can be used for sequencing/partitioning
- Partition in space lets a few states be “seen at the same time”
Implications: Systems, Algorithms and Visualizations

- Many computations can be made progressive (CALM)
- Monotonic = easier to visualize & understand

- Time and Space can be used to organize independent things
  - Even if they’re progressive

- Some things are truly sequenced
  - The classic: state mutation in time
    - Though this is often artificial
  - Exponential problems
Outline

1. Perspective
2. Interactive as Distributed
3. Async Interaction
4. CALM Progress
5. More Progress
What About Approximation?

- Where is the monotonicity?
  - Count
  - Average?
  - $\varepsilon$
Confidence Bounds for Average

- Hoeffding:
  \[ \epsilon_n = (b - a) \left( \frac{1}{2n} \ln \left( \frac{2}{1 - p} \right) \right)^{1/2} \]

- CLT-based:
  \[ \epsilon_n = \left( \frac{z_p^2 T_{n,2}(v)}{n} \right)^{1/2} \]
More Hints

- Sub/Super-martingales
- Monotonicity of Expectation
- “Stochastic CALM”
Questions/Challenges I: End-to-End Progressive

**Consistent Progressive Perception**
- Establish the notion of "consistency" between human and computational models
- Formalize the connection between perception, monotonicity and coordination

**What needs to be Progressive?**
- Coordination-free systems
- Monotonicity of approximation
- Monotonicity of user experience
Questions/Challenges II

Pragmatics

What tasks merit progressive feedback?

Separately, what tasks merit progressive approximation?

Interaction and Control Loops

When does user input suggest starting “a new session” (a clock tick)?

How does the biased human input channel interact with approximation rigor?

Are humans more likely to perform truly non-monotone tasks, and should we support that explicitly?
Takeaways

- Consider Systems, Statistics and UX
- Online Results, Aggregations:
  - A special case of streaming computation
- HCI is a Distributed System
  - Worry about consistency, reordering, latency variance
- CALM makes things *much* easier
  - Monotonicity implies coordination-freeness
  - At system, stats and UX levels
Citations


Citations, Cont.

The First Online Aggregation UI

- Continuous feedback
  - approximation
  - confidence
  - progress
- Ongoing control of sampling

With thanks to Andrew MacBride, 1996