Behavioral Simulations in MapReduce

Guozhang Wang, Cornell University

with Marcos Vaz Salles, Benjamin Sowell, Xun Wang, Tuan Cao, Alan Demers, Johannes Gehrke, and Walker White

MSR DMX August 20, 2010
What are Behavioral Simulations?

- Simulations of individuals that interact to create emerging behavior in complex systems

- Application Areas
  - Traffic
  - Ecology
  - Sociology
  - etc
Why Behavioral Simulations?

• Traffic
  – Congestion cost $87.2 billion in the U.S. in 2007
  – More people killed by air pollution than accidents
  – Detailed models: micro-simulators not scale to NYC!

• Ecology
  – Hard to scale to large fish schools or locust swarms
Challenges of Behavioral Simulations

• **Easy to program → not scalable**
  – Examples: Swarm, Mason

• **Scalable**
  – Examples: TRANSIMS, DynaMIT (traffic), GPU implementation of fish simulation (ecology)
  – Hard-coded models, compromise level of detail

Can we do better?
Our Contribution

• A new simulation platform that combines:
  – Ease of programming
  • Program simulations in time-stepped pattern
  • *BRASIL*: Scripting language for domain scientists
  – Scalability
  • Execute simulations in a dataflow model
  • *BRACE*: Special-purpose MapReduce engine
Talk Outline

• Motivation

• Ease of Programming
  – Program Simulations in Time-stepped Pattern
  – BRASIL

• Scalability
  – Execute Simulations in Dataflow Model
  – BRACE

• Experiments

• Conclusion
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005

- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish
A Running Example: Fish Schools

- **Time-stepping**: agents proceed in ticks
- **Concurrency**: agents are concurrent within a tick
- **Interactions**: agents continuously interact
- **Spatial Locality**: agents have limited visibility and reachability
Classic Solutions for Concurrency

- Preempt conflicts $\rightarrow$ locking
- Rollback in case of conflicts $\rightarrow$ optimistic concurrency control

Problems:
- Strong iterations $\rightarrow$ many conflicts
  - Either lots of lock contention
  - Or lots of rollbacks
- Not scale well
State-Effect Pattern

- Programming pattern to deal with concurrency
- Follows time-stepped model
- **Core Idea**: Make all actions inside of a tick order-independent
States and Effects

• States:
  – Snapshot of the agent status at the beginning of the tick

• Effects:
  – Intermediate results *per agent* as a result of interaction, used to calculate new states
States and Effects of Fish Agents

• **States:**
  – Position, velocity vector

• **Effects:**
  – Sets of attraction forces from other fish
  – Sets of avoidance forces from other fish
Two Phases of a Tick

• Query: capture agent interaction
  – Read states \(\rightarrow\) write effects
  – Each effect set is associated with \textit{combinator} function
  – Effect writes are \textit{order-independent}

• Update: refresh world for next tick
  – Read effects \(\rightarrow\) write states
  – Reads and writes are totally local
  – State writes are \textit{order-independent}
A Tick in Fish Simulation

• Query
  – For fish f in visibility $\alpha$:
    • Write repulsion to f’s effects
  – For fish f in visibility $\rho$:
    • Write attraction to f’s effects

• Update
  – new velocity = combined repulsion + combined attraction + old velocity
  – new position = old position + old velocity
A Tick in Fish Simulation

• Query
  – For fish $f$ in visibility $\alpha$:
    • Write repulsion to $f$’s effects
  – For fish $f$ in visibility $\rho$:
    • Write attraction to $f$’s effects

• Update
  – new velocity = combined repulsion + combined attraction + old velocity
  – new position = old position + old velocity
• High-level language for domain scientists

• Object-oriented style

• Programs specify behavior logic of individual agents

• Syntax enforces state-effect pattern
class Fish {
    // The fish location & vel (x)
    public state float x : (x+vx); #range[-1,1];
    public state float vx : vx + rand() + avoidx / count * vx;
    // Used to update our velocity (x)
    private effect float avoidx : sum;
    private effect int count : sum;
    ...
}
class Fish {
    // The fish location & vel (x)
    ...

    /** The query-phase for this fish. */
    public void run() {
        // Use "forces" to repel fish too close
        foreach (Fish p : Extent<Fish>) {
            p.avoidx <- 1 / abs(x - p.x);
            ...
            p.count <- 1;
        }
    }
}
Compiling BRASIL

• BRASIL translates to Monad Algebra

```java
foreach (Fish p : Extent<Fish>) {
    p.avoidx = -1 / x - p.x;
}
```

```sql
Select id, sum(1/f1.x - f2.x), count(*)
from Fish f1, Fish f2
where f1.x < f2.x + 1
and f1.x > f2.x - 1
``` 

• Can reuse classic DB optimization techniques – indexing, join reordering

• Details of translation in our VLDB 2010 paper
Talk Outline

• Motivation

• Ease of Programming
  – Program Simulations in Time-stepped Pattern
  – BRASIL

• Scalability
  – Execute Simulations in Dataflow Model
  – BRACE

• Experiments

• Conclusion
How to Scale to Millions of Fish?

- Use multiple nodes in a cluster of machines for large simulation scenarios
- Need to efficiently parallelize state-effect when agents are partitioned across nodes
  - One query phase as a spatial join
  - Simulation as iterated spatial joins
  - Parallelize iterated spatial joins as a dataflow in MapReduce
BRACE (Big Red Agent Computation Engine)

• Special-purpose MapReduce engine for behavioral simulations

• Basic Optimizations
  – Keep data in main memory
  – Do Not checkpoint every iteration

• Optimizations based on Spatial Properties:
  – Collocate tasks
  – Minimize communication overhead
Spatial Partitioning

- Partition simulation space into regions, each handled by a separate node
Communication Between Partitions

- **Visible Region**: not owned, but need to be seen for the query phase

- Communicate to replicate agents, and send **non-local** effects

- Communicate only with neighbors
State-Effect with Partitioned Agents

- Communicate new states before next tick

Diagram:

- Tick
  - Query: state → effects
  - Communicate Effects
    - Update: effects → new state
    - Communicate New State

Legend:
- Green: Owned
- Blue: Visible
State-Effect with Partitioned Agents

• Communicate non-local effects before update
From State-Effect to Map-Reduce

- **Tick**
  - Query: state $\rightarrow$ effects
  - Communicate Effects
  - Update: effects $\rightarrow$ new state
  - Communicate New State

- **Map**
  - $Map_1 \ t$
  - $Reduce_1 \ t$
  - $Map_2 \ t$
  - $Reduce_2 \ t$
  - $Map_1 \ t+1$

- **Processes**
  - Distribute data
  - Assign effects (partial)
  - Forward data
  - Aggregate effects
  - Update Redistribute data
Effect Inversion

- If only have local effects, each tick will only have one communication round
Effect Inversion Is Always Possible

• **Theorem**: Every behavioral simulation written in BRASIL that uses non-local effects can be rewritten to an equivalent simulation that uses local effects only
  
  — Proof in the VLDB 2010 paper
Intuition of Effect Inversion Theorem

Non-local Effect Writes

Non-local State Reads + Local Effect Writes

α → 2α
Talk Outline

• Motivation

• Ease of Programming
  – Program Simulations in Time-stepped Pattern
  – BRASIL

• Scalability
  – Execute Simulations in Dataflow Model
  – BRACE

• Experiments

• Conclusion
Experimental Setup

• BRACE prototype
  – Grid partitioning
  – KD-Tree spatial indexing, rebuild every tick
  – Basic load balancing
  – No checkpointing

• Hardware: Cornell WebLab Cluster (60 nodes, 2xQuadCore Xeon 2.66GHz, 4MB cache, 16GB RAM)
Implemented Simulations

• Traffic Simulation
  – Best-effort reimplementation of MITSIM lane changing and car following
  – Large segment of highway

• Bacteria Simulation
  – Simple artificial society simulation

• Fish School Simulation
  – Model of collective animal motion by Couzin et al., Nature, 2005
Scalability: Traffic

- Nearly linear scalability; notch consequence of Web Lab’s IP routing / multi-switch architecture
Effect Inversion: Bacteria

- 16-node with indexing and effect inversion
- 10,000 epochs of bacteria simulation
• 16-node with load balancing turned on
• Fish simulation of two independent schools that swim in opposite directions
• Scientists want to run their simulations in the cloud

• Can we use the cloud?
  – Large and variable latency $\rightarrow$ latency compensation techniques
  – Large number of unreliable nodes $\rightarrow$ low-overhead checkpoints
  – Money as new optimization metric $\rightarrow$ think about how to allocate tasks
Conclusions

• Behavioral Simulations can have huge impact, but need to be run at large-scale

• New programming environment for behavioral simulations
  – *Easy to program*: Simulations in the state-effect pattern → BRASIL
  – *Scalable*: State-effect pattern in special-purpose MapReduce Engine → BRACE

• BRACE is coming to the cloud!
  – And we are going to simulate NYC 😊
Backup Slides
MITSIM: Single-Node Traffic Simulator

Stockholm, Norway
An Observation about Parallelism

• A query phase in one tick is a spatial join

<table>
<thead>
<tr>
<th>Id</th>
<th>X</th>
<th>Y</th>
<th>vX</th>
<th>vY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
<td>1.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td>1.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2.3</td>
<td>2.4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Behavioral simulations are large, iterated spatial joins

• *We can parallelize iterated spatial joins in MapReduce!*
From State-Effect to MapReduce

Tick

Query

Update

\[ \text{Map}_t \]

\[ \text{Reduce}_t \]

\[ \text{Map}_{t+1} \]

- Update each agent as a key-value pair; load agents when \( t = 0 \)
- Process spatial joins and output effects
- Update each agent as a key-value pair