Accelerating Parallel Analysis of Scientific Simulation Data via Zazen

Tiankai Tu, Charles A. Rendleman, Patrick J. Miller, Federico Sacerdoti, Ron O. Dror, and David E. Shaw

D. E. Shaw Research
Motivation

- **Goal:** To model biological processes that occur on the millisecond time scale
- **Approach:** A specialized, massively parallel supercomputer called *Anton* (2009 ACM Gordon Bell Award for Special Achievement)
### Millisecond-scale MD Trajectories

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A biomolecular system:</td>
<td>25 K atoms</td>
</tr>
<tr>
<td>Position and velocity:</td>
<td>24 bytes/atom</td>
</tr>
<tr>
<td><strong>Frame size:</strong></td>
<td><strong>0.6 MB/frame</strong></td>
</tr>
<tr>
<td>Simulation length:</td>
<td>$1 \times 10^{-3}$ s</td>
</tr>
<tr>
<td>Output interval:</td>
<td>$10 \times 10^{-12}$ s</td>
</tr>
<tr>
<td><strong>Number of frames:</strong></td>
<td><strong>100 M frames</strong></td>
</tr>
</tbody>
</table>
Part I: How We Analyze Simulation Data in Parallel
An MD Trajectory Analysis Example: Ion Permeation
A Hypothetic Trajectory

20,000 atoms in total; two ions of interest

Ion A

Ion B
Ion State Transition

- Above channel
- Into channel from above
- Into channel from below
- Inside channel
- Below channel

S
Typical Sequential Analysis

- Maintain a main-memory resident data structure to record states and positions
- Process frames in ascending simulated physical time order

**Strong inter-frame data dependence:**
Data analysis tightly coupled with data acquisition
Problems with Sequential Analysis

- Millisecond-scale trajectory size: 60 TB
- Local disk read bandwidth: 100 MB/s
- Time to fetch data to memory: 1 week
- Analysis time: Varied
- Time to perform data analysis: Weeks

Sequential analysis lack the computational, memory, and I/O capabilities!
A Parallel Data Analysis Model

Specify which frames to be accessed

Decouple data acquisition from data analysis

- Trajectory definition
- Stage 1: Per-frame data acquisition
- Stage 2: Cross-frame data analysis
Trajectory Definition

Every other frame in the trajectory

![Graph showing ion trajectories](attachment:graph.png)

- Ion A
- Ion B
Per-frame Data Acquisition (stage 1)
Cross-frame Data Analysis (stage 2)

Analyze ion A on P0 and ion B on P1 in parallel
Inspiration: Google’s MapReduce

Google File System

Input files

map(...) K1: \{v1\} K2: \{v2\}

map(...) K1: \{v1\} K2: \{v2\}

map(...) K1: \{v1_k\} K2: \{v2_k\}

Output file

Output file

K1: \{v1_j, v1_i, v1_k\}

reduce(K1, ...)

K2: \{v2_k, v2_j, v2\}

reduce(K2, ...)

Trajectory Analysis Cast Into MapReduce

- Per-frame data acquisition (stage 1): map()
- Cross-frame data analysis (stage 2): reduce()
- Key-value pairs: connecting stage 1 and stage 2
  - Keys: categorical identifiers or names
  - Values: including timestamps
  - Examples: $(\text{ion}_i, \text{id}_j, (t_k, \text{x}_{ik}, \text{y}_{jk}, \text{z}_{jk}))$

Key

Value
The HiMach Library

- **A MapReduce-style API** that allows users to write Python programs to analyze MD trajectories

- **A parallel runtime** that executes HiMach user programs in parallel on a Linux cluster automatically

- **Performance results on a Linux cluster:**
  - 2 orders of magnitude faster on 512 cores than on a single core
Typical Simulation–Analysis Storage Infrastructure

- Parallel supercomputer
- File servers
- I/O node
- Parallel analysis programs
- Local disks
- Analysis nodes
Part II: How We Overcome the I/O Bottleneck in Parallel Analysis
Trajectory Characteristics

- A large number of small frames
- Write once, read many
- Distinguishable by unique integer sequence numbers
- Amenable to out-of-order parallel access in the map phase
Our Main Idea

- At simulation time, actively cache frames in the local disks of the analysis nodes as the frames become available.

- At analysis time, fetch data from local disk caches in parallel.
Limitations

- Require large aggregate disk capacity on the analysis cluster
- Assume relatively low average simulation data output rate
An Example

Analysis node 0
- Merged bitmap: 1 1 1 1
- Remote bitmap: 0 1 0 1
- Local bitmap: 1 0 1 0

Analysis node 1
- Merged bitmap: 1 1 1 1
- Remote bitmap: 1 0 1 0
- Local bitmap: 0 1 0 1

NFS server

Diagram showing the analysis nodes with their respective bitmaps and file structures.
How to guarantee that each frame is read by one and only one node in the face of node failure and recovery?

The Zazen Protocol
The Zazen Protocol

- Execute a distributed consensus protocol before performing actual disk I/O
- Assign data retrieval tasks in a location-aware manner
- Read data from local disks if the data are already cached
- Fetch missing data from file servers
- No metadata servers to keep record of who has what
The Zazen Protocol (cont’d)

- **Bitmaps**: a compact structure for recording the presence or non-presence of a cached copy
- **All-to-all reduction algorithms**: an efficient mechanism for inter-processor collective communications (used an MPI library in practice)
Implementation

- The Bodhi library
- The Bodhi server
- The Zazen protocol
Performance Evaluation
Experiment Setup

- A Linux cluster with 100 nodes
- Two Intel Xeon 2.33 GHz quad-core processors per node
- Four 500 GB 7200-RPM SATA disks organized in RAID 0 per node
- 16 GB physical memory per node
- CentOS 4.6 with a Linux kernel of 2.6.26
- Nodes connected to a Gigabit Ethernet core switch
- Common accesses to NFS directories exported by a number of enterprise storage servers
Fixed-Problem-Size Scalability

Execution time of the Zazen protocol to assign the I/O tasks of reading 1 billion frames
Fixed-Cluster-Size Scalability

Execution time of the Zazen protocol on 100 nodes assigning different number of frames
Efficiency I: Achieving Better I/O BW

One Bodhi daemon per user process

One Bodhi daemon per analysis node

GB/s

Application read processes per node

GB/s

Application read processes per node

1-GB
256-MB
64-MB
2-MB
Efficiency II: Comparison w. NFS/PFS

- NFS (v3) on separate enterprise storage servers
  - Dual quad-core 2.8-GHz Opteron processors, 16 GB memory, 48 SATA disks organized in RAID 6
  - Four 1 GigE connection to the core switch of the 100-node cluster

- PVFS2 (2.8.1) on the same 100 analysis nodes
  - I/O (data) server and metadata server on all nodes
  - File I/O performed via the PVFS2 Linux kernel interface

- Hadoop/HDFS (0.19.1) on the same 100 nodes
  - Data stored via HDFS’s C library interface, block sizes set to be equal to file sizes, three replications per file
  - Data accessed via a read-only Hadoop MapReduce Java program (with a number of best-effort optimizations)
Efficiency II: Outperforming NFS/PFS

I/O bandwidth of reading files of different sizes

File size for read

- NFS
- PVFS2
- Hadoop/HDFS
- Zazen

GB/s

2 MB 64 MB 256 MB 1 GB
Efficiency II: Outperforming NFS/PFS

Time to read one terabyte of data

- NFS
- PVFS2
- Hadoop/HDFS
- Zazen

Time (s)

File size for read

- 2 MB
- 64 MB
- 256 MB
- 1 GB
Read Perf. under Writes (1GB/s)

File size for writes
- No writes
- 1 GB files
- 256 MB files
- 64 MB files
- 2 MB files

Normalized performance vs File size for reads
End-to-End Performance

- A HiMach analysis program called *water residence* on 100 nodes
- 2.5 million small frame files (430 KB each)
Robustness

- Worst case execution time is $T(1 + \delta \frac{B}{b})$
- The water-residence program re-executed with varying number of nodes powered off
Summary

Zazen accelerates order-independent, parallel data access by (1) actively caching simulation output, and (2) executing an efficient distributed consensus protocol.

- Simple and robust
- Scalable on a large number of nodes
- Much higher performance than NFS/PFS *
- Applicable to a certain class of time-dependent simulation datasets *