JTool: Accessing Warehoused Collections of Objects with Java

S. Bailey
Laboratory for Advanced Computing
University of Illinois at Chicago

R. L. Grossman
Laboratory for Advanced Computing
University of Illinois at Chicago
and Magnify, Inc.

Overview

- Introduction
- Related Work
- Using JTool
- Design and Implementation
- Experimental Results
- Summary and Conclusions

What is Data Mining?

- Data mining is the automatic discovery of patterns, associations, anomalies, and changes in large amounts of operational and transactional data.
- Emphasis on discovery (search) vs validation of patterns
- Two paths
  - patterns used to improve predictive models (PM)
  - patterns used to gain insight into data (KD)

Object Warehouses

- Object warehouses
  - data repositories which support the analysis of large collections of objects
  - compromise safety for performance
  - precompute as much as makes sense
  - index as much as makes sense
  - a great deal of research on data warehouses for relational data, much less for object data
  - need for wide area object warehouses

Goal

- Develop a software tool for creating data warehouses of Java objects for data mining
  - applications to data preparation and data cleaning
  - applications to wide area data mining and meta-learning
  - we have a working prototype: JTool (Version 0.2)

Example 1: Detecting Network Intrusions

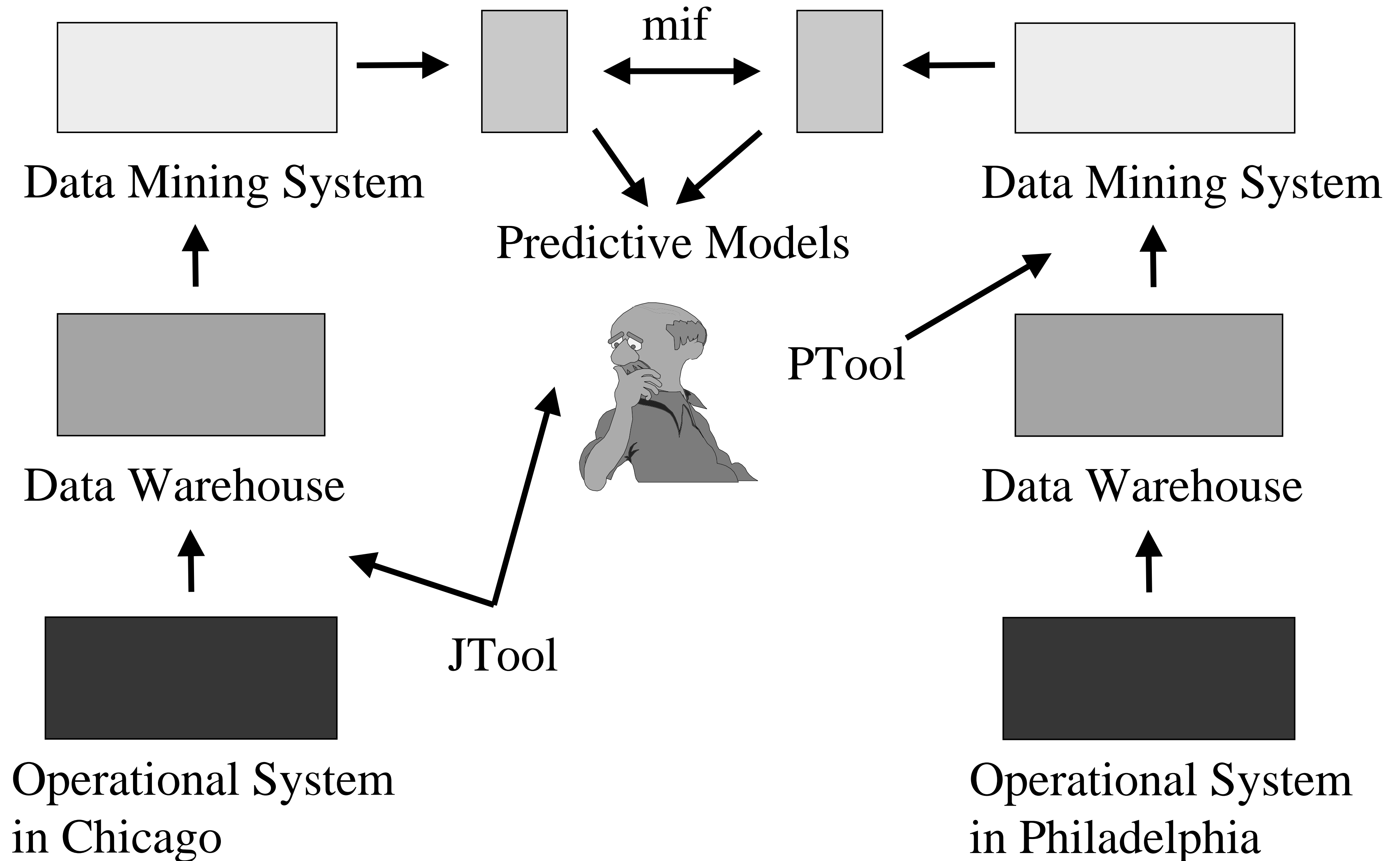
- Create audit and security logs
  - number of logins
  - number of incorrect logins
  - number of telnets
  - number of file accesses
- Look for patterns predictive of intrusions or other misuse

Example 2:
Detecting Credit Card Fraud

[0] Account number:	4500089050201002
[1] Transaction Amount:	45.06
[2] Timestamp:	95214013238
[3] Acquiring Bank:	840
[4] Issuing Bank:	124
[5] Store Type:	5310
[6] Velocity 1:	-0.795767
[7] Velocity 2:	-0.609230

plus 75 additional attributes

Example 3: Wide Area Data Mining



Model Interchange Format

<attribute-list>

<li type=data-attribute attribute-number=1> card-number

<li type=derived attribute-number=8> velocity-three-hours

</attribute-list>

<logistic-regression weight = 0.3>

<li coefficient=0.239494> velocity three hours

<li coefficient=0.495858> merchant code

</logistic-regression>

<decision-tree weight = 0.7>

<li decision-attribute=6 threshold= 0.459507 left-node=12 right-node=18> velocity three hours

Related Work

- Directly supporting orthogonal persistence
  - changing the Java compiler or changing the Java virtual machine
  - preprocessing the Java source code, or post-processing the Java byte code
- Interfaces to existing databases
  - JDBC, ODMG, JRB, JUB,
  - legacy systems ...
- Interface to file systems
  - flattening, streaming, ...

The Persistence Model

- Principles well understood (by the folks in this room):
- Persistence should be orthogonal to type
- Determining which data persists
  - Persistence by attachment or reachability
  - Explicit persistence

The spirit was willing, but the flesh was weak: currently, we do not support orthogonal persistence, persistence by reachability, ...

A Spectrum for Data Management Systems

Databases	Data Warehouses
• transactions environments • many writes • updates of simple, normalized, relational data	• analysis environments • many reads • precomputing and indexing of summary and derived data • complex queries on complex objects

---

Spectrum: Safe updates, Safe appends, Safe checkpoints, Safe reads

JTool Design Goals

- Lightweight; optimized for read-only queries, precomputing, and indexing

- Scalable to large collections of objects; objects containing large numbers of attributes; and to queries which are numerically intensive

Core API - Objects and Utilities

Objects	Methods
Ref	Ref() Object Deref() void Persist()
Store	Store(String) void insert_element(Ref) void Next(Ref) void Reset()
JTool	static Ref New(Store, Object) static FinalizeJTool()

Utilities	Usage
JTool_Register	JTool_Register <Object List>

A Simple Population Program for "Event" Objects

```
public static void main( String args[] )
{

  Store store = new Store( "PsiEvents" );

  Ref eventRef;

  try{
    for(  int a = 0 ; a < 1000 ; a++ )
    {
          eventRef = JTool.New(  s, new Event( s ) );
          store.insert_element( eventRef );
    }
    FinalizeJTool();
  }catch(Exception e )
   { System.out.println( "Exception: " + e)};
}
```

A Simple Access Program for “Event” Objects

```
public static void main( String args[] )
{

  Store store = new Store( ”PsiEvents" );

  Ref eventRef = new Ref();

  Event event;
  try{
     while(  s.Next( eventRef ) )
     {
      event = (Event)eventRef.Deref();
      System.out.println( event.vertex );
     }
  }catch(Exception e){};
  FinalizeJTool();
}
```

Rules for the Programmer

- All persistent classes must implement Serializable.
- All persistent classes must be *registered,* with the JTool_Register utility, *after* compile time (i.e. third party, compiled classes are supported if they implement Serializable).
- All atomic objects must be smaller than the Segment size (currently 65K).
- All references between atomic objects must be of type Ref.

Design and Implementation

- Physical Storage Management
  - JTool manages a hierarchy of physical extents
  - stores, folios, segments, ...
- Utilize Object Serialization
  - Currently, JTool uses the Object Serialization support in JDK 1.1.1 to facilitate the casting of bytes held in the JTool.

Physical Storage Management

- *Name Space:* a collection of stores with a common name space. Objects can refer to other objects within the name space.
- *Store:* a collection of folios
- *Folio:* a data file of objects which is divided into smaller units called segments. Folios are the basic unit managed by the storage or file system
- *Segment:* collection of contiguous objects. Segments are the basic units which are retrieved from disk and written into memory when a persistent object is dereferenced.

The Use of Object Serialization

- JDK 1.1 Object Serialization was a very easy way to cast atomic objects to byte streams and vice versa.
- Unlike traditional Object Serialization, object trees which use JTool Refs do not automatically deferenced the entire tree when the root is dereferenced.
- The use of Refs and Name Spaces allows users to place objects in a 64bit randomly accessed address space with some control over object locality.
- Object Serialization introduces translation inefficiencies which need to be overcome.

The Ref

Refs point to objects in the persistent space. When the `Deref()` method is called or a `JTool, New()` is invoked, an instance of Ref also points to a transient image of the persistent object.

```
class Ref implements Serializable
{
        transient Object object;
        long ppointer;
         ...
}
```

The Waddle

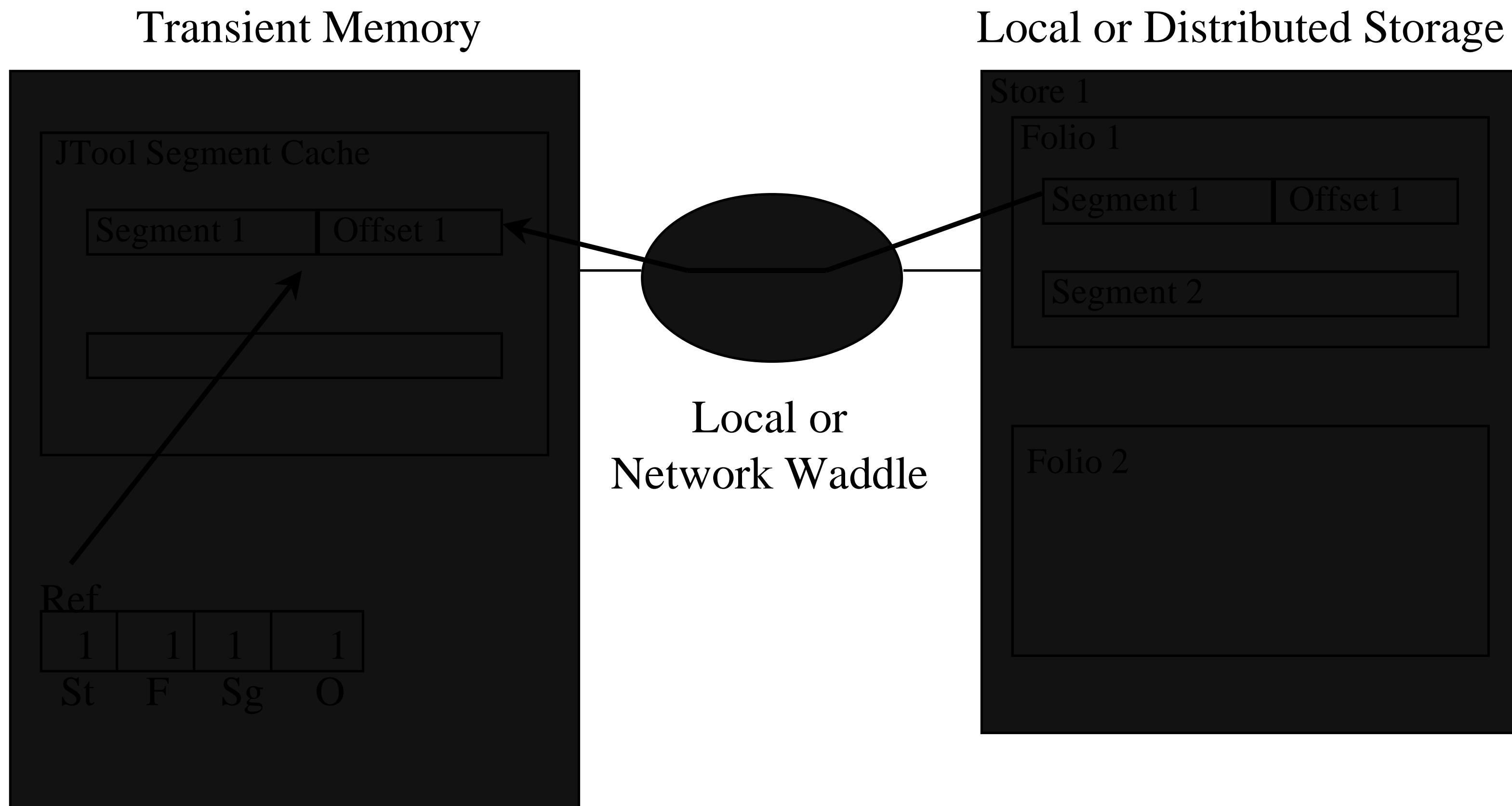
- The Waddle objects of JTool is responsible for the actual fetching of segments from secondary storage into the JTool cache.

- Currently, only a local file Waddle is provided with JTool.

- Network, compressed file, and tape Waddles will be available for future implementations of JTool.

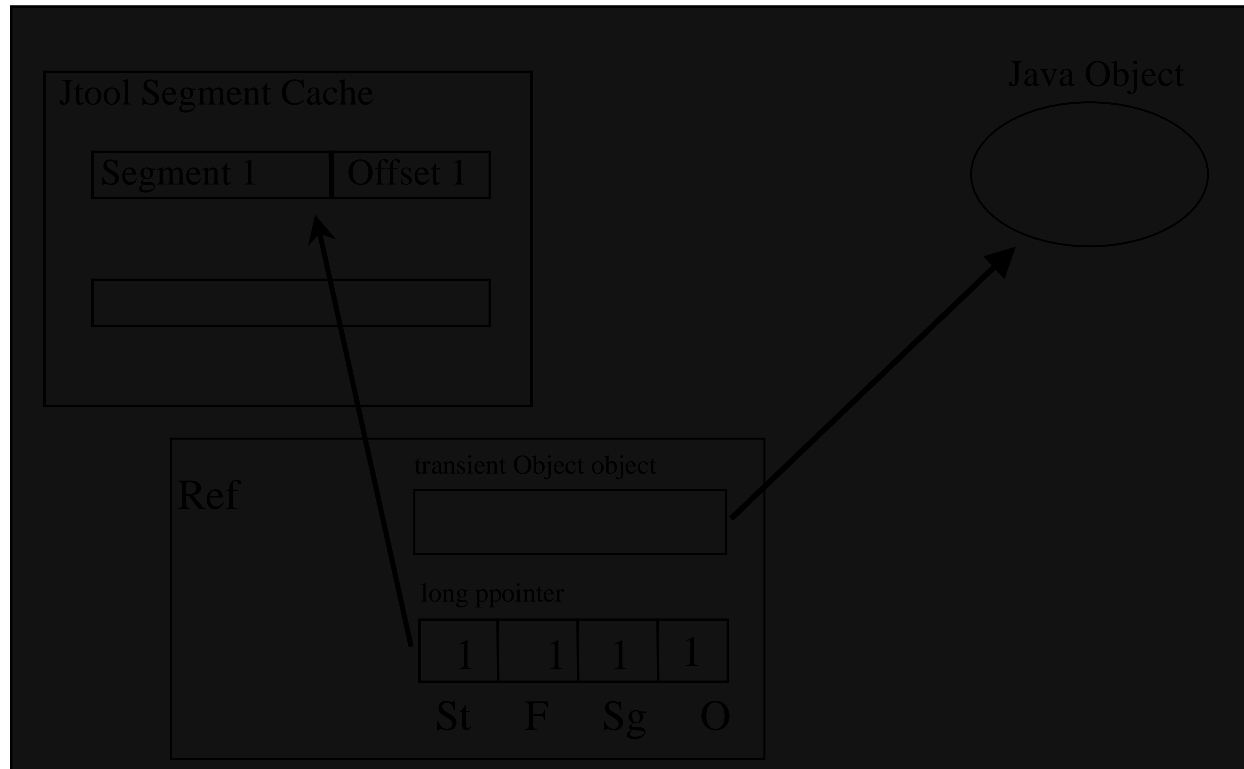
- Network waddles will allow direct Store access by applets.

De-referencing a Ref



De-referencing a Ref Cont.

Transient Memory



Experimental Results

- Experiments were conducted on a Sun Sparc 20 with 32 Mbytes of RAM running Solaris 2.5.1 and Jdk 1.1.1.

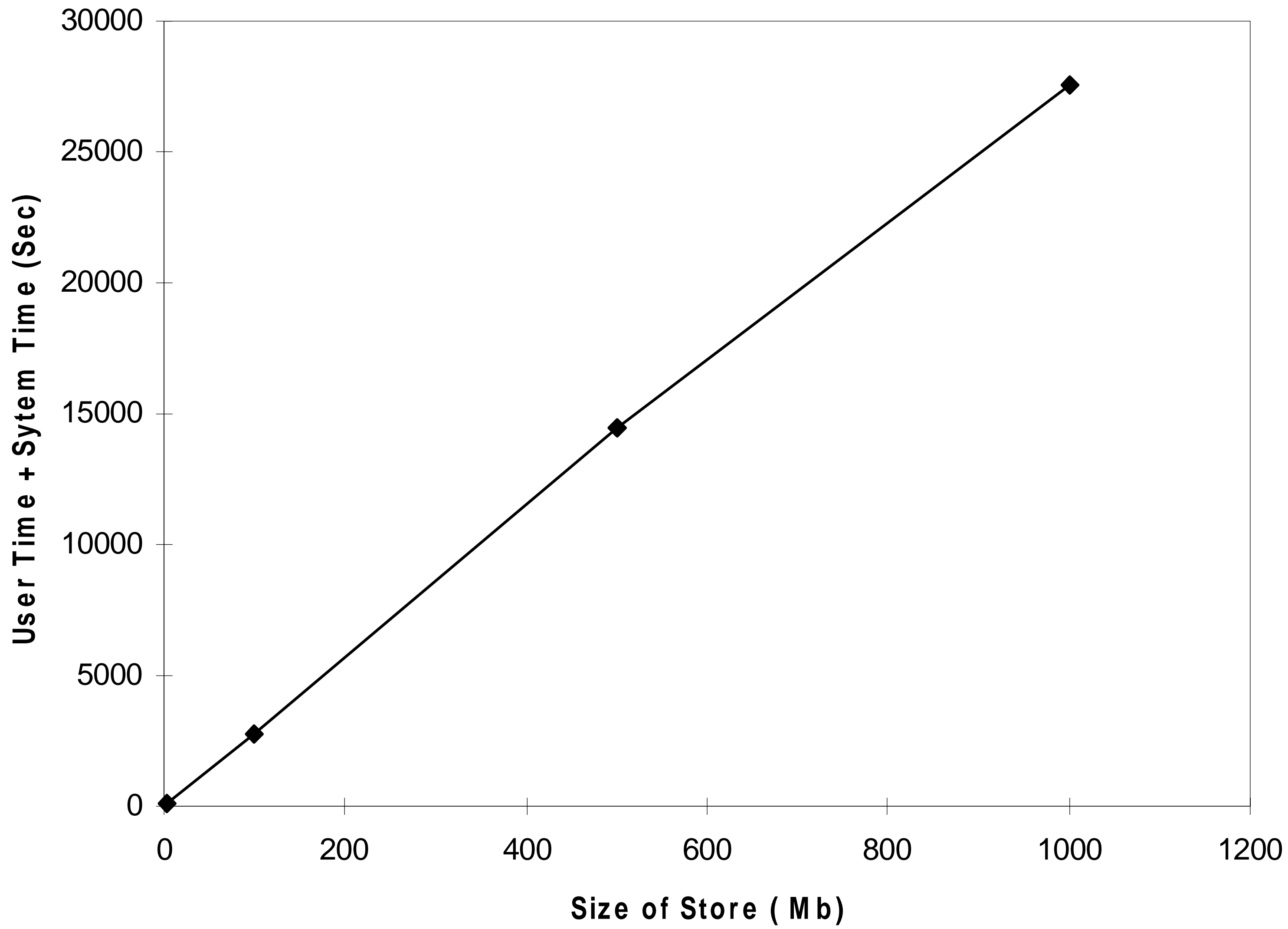
- The disk on which we populated stores was a 9 Gig Seagate NFS mounted to the SparcStation over standard ethernet.

- For ease of comparison with our past work mining scientific data, the data set contained synthetic data called Events, broadly similar to data arising in high energy physics.

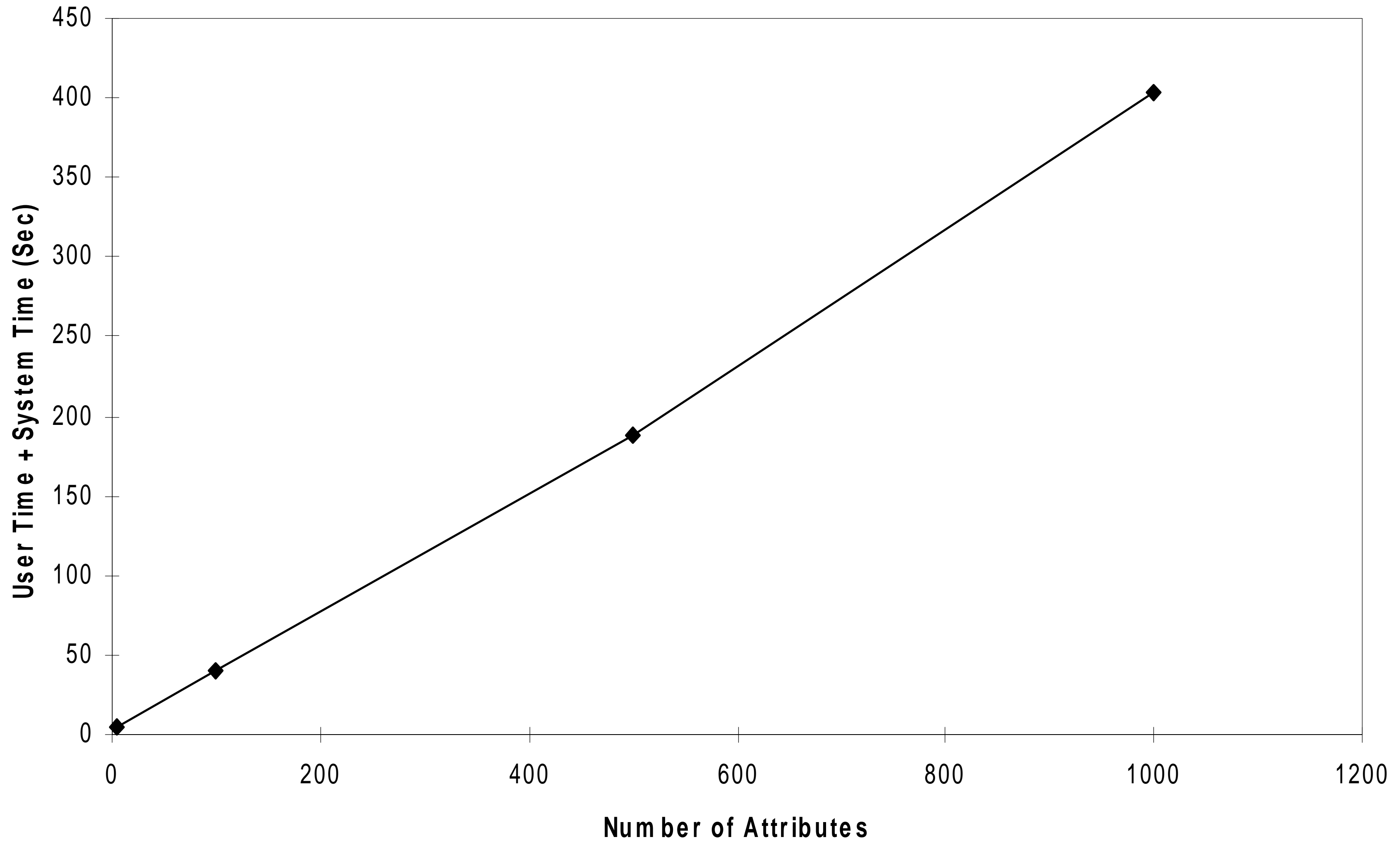
Tests

1. Access time for examining all entire Event objects (i.e. including its Leptons and Jets) in a store as the Store size varied.

2. Access time for examining all entire Event objects in a store while varying the number of attributes per Event.



Access Time vs. Size of Store
30000
25000
20000
15000
10000
5000
0
User Time + Sytem Time (Sec)
0
200
400
600
800
1000
1200
Size of Store (Mb)


Access Time vs. Number of Attributes
User Time + System Time (Sec)
450
400
350
300
250
200
150
100
50
0
0
200
400
600
800
1000
1200
Number of Attributes