Active Disks - Remote Execution for Network-Attached Storage

Erik Riedel

Thesis Defense
Electrical and Computer Engineering

Prof. David Nagle, ECE
Prof. Christos Faloutsos, SCS
Prof. Garth Gibson, SCS
Prof. Pradeep Khosla, ECE
Dr. Jim Gray, Microsoft Research
A number of important I/O-intensive applications can take advantage of *computational power* available directly at storage devices.

**Computation in Storage**
A number of important I/O-intensive applications can take advantage of computational power available directly at storage devices to improve their overall performance.
A number of important I/O-intensive applications can take advantage of *computational power* available directly at storage devices to improve their overall *performance*, more effectively balance their consumption of system-wide *resources*
A number of important I/O-intensive applications can take advantage of *computational power* available directly at storage devices to improve their overall *performance*, more effectively balance their consumption of system-wide *resources*, and provide *functionality* that would not otherwise be available.

**Computation in Storage**

**Performance Model**

**Applications & Prototype**

**Drive-Specific Functionality**
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Motivation

Allow faster, more flexible access to storage

Would you say your storage requirements are . . .

- Increasing Very Rapidly/Rapidly: 84%
- Increasing Slowly: 15%
- Not Increasing: 1%

Storage requirements are pushing
- more data
- increased sharing
- richer data types
- novel applications

Which of the following applications or activities are contributing to this growth?

- New On-Line Applications: 68%
- Data Warehousing: 67%
- Internet/Intranet: 62%
- On-Line Archival Information: 52%
- Increasing Size of Customer Records: 47%
- Data Consolidation: 47%
- Year 2000: 41%
- Electronic Commerce: 39%
- Windows NT Implementation: 35%
- Enterprise Resource Planning (ERP): 34%
- Mergers and Acquisitions: 33%

data from www.EMC.com survey of Senior IS Executives
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Evolution of Disk Drive Electronics

Integration

- reduces chip count
- improves reliability
- reduces cost
- future integration to processor on-chip
- but there must be at least one chip
Higher and higher levels of integration in electronics

- specialized drive chips combined into single ASIC
- technology trends push toward integrated control processor
- Siemens TriCore - 100 MHz, 32-bit superscalar today
  - to 500 MIPS within 2 years, up to 2 MB on-chip memory
- Cirrus Logic 3CI - ARM7 core today
  - to ARM9 core at 200 MIPS in next generation

High volume, commodity product

- 145 million disk drives sold in 1998
  - about 725 petabytes of total storage
  - manufacturers looking for value-added functionality
Opportunity

TPC-D 300 GB Benchmark, Decision Support System

Digital AlphaServer 8400
- 12 x 612 MHz 21164
- 8 GB memory
- 3 64-bit PCI busses
- 29 FWD SCSI controllers

= 7,344 total MHz

= 104,000 total MHz
(with 200 MHz drive chips)

= 5,200 total MB/s
(at 10 MB/s per disk)

Storage
- 520 rz29 disks
- 4.3 GB each
- 2.2 TB total

= 3 x 266 = 798 MB/s

= 29 x 40 = 1,160 MB/s

Database Server
Active Disks execute application-level code on drives

Basic advantages of an Active Disk system

- parallel processing - lots of disks
- bandwidth reduction - filtering operations are common
- scheduling - little bit of “strategy” can go a long way

Characteristics of appropriate applications

- execution time dominated by data-intensive “core”
- allows parallel implementation of “core”
- cycles per byte of data processed - computation
- data reduction of processing - selectivity
Example Application

Data mining - association rules [Agrawal95]
  • retail data, analysis of “shopping baskets”
  • frequent sets summary counts
  • count of 1-itemsets and 2-itemsets
  • milk & bread => cheese
  • diapers & beer

Partitioning with Active Disks
  • each drive performs count of its portion of the data
  • counts combined at host for final result
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Performance Model

Application Parameters

\( N_{\text{in}} = \) number of bytes processed

\( N_{\text{out}} = \) number of bytes produced

\( w = \) cycles per byte

\( t = \) run time for traditional system

\( t_{\text{active}} = \) run time for active disk system

\( d = \) number of disks

System Parameters

\( s_{\text{cpu}} = \) CPU speed of the host

\( r_d = \) disk raw read rate

\( r_n = \) disk interconnect rate

Active Disk Parameters

\( s_{\text{cpu}}' = \) CPU speed of the disk

\( r_d' = \) active disk raw read rate

\( r_n' = \) active disk interconnect rate

Traditional vs. Active Disk Ratios

\( \alpha_N = \frac{N_{\text{in}}}{N_{\text{out}}} \)

\( \alpha_d = \frac{r_d'}{r_d} \)

\( \alpha_n = \frac{r_n'}{r_n} \)

\( \alpha_s = \frac{s_{\text{cpu}}'}{s_{\text{cpu}}} \)
Performance Model

Traditional server:

\[ t = \max \left( \frac{N_{\text{in}}}{d \cdot r_d}, \frac{N_{\text{in}}}{r_n}, \frac{N_{\text{in}} \cdot w}{s_{\text{cpu}}} \right) + (1 - p) \cdot t_{\text{serial}} \]

Active Disks:

\[ t_{\text{active}} = \max \left( \frac{N_{\text{in}}}{d \cdot r_d'}, \frac{N_{\text{out}}}{r_n'}, \frac{N_{\text{in}} \cdot w}{d \cdot s_{\text{cpu}}'} \right) + (1 - p) \cdot t_{\text{serial}} \]
Throughput Model

Scalable throughput

- speedup = (#disks)/(host-cpu-speed/disk-cpu-speed)
Scalable throughput

- speedup = (#disks)/(host-cpu-speed/disk-cpu-speed)
Scalable throughput

- **speedup** = (#disks)/(host-cpu-speed/disk-cpu-speed)
- (host-cpu/disk-cpu-speed) ~ 5  (two processor generations)
Scalable throughput

- **speedup** = (#disks)/(host-cpu-speed/disk-cpu-speed)
- (host-cpu/disk-cpu-speed) ~ 5 (two processor generations)
- **selectivity** = #bytes-input / #bytes-output
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Prototype Comparison

Traditional System

Digital AlphaServer 500/500
- 500 MHz, 256 MB memory
- Seagate Cheetah disks
- 4.5 GB, 11.2 MB/s

Digital AXP 3000/400 "Active Disks"
- 133 MHz, 64 MB, software NASD
- Seagate Medallist disks
- 4.1 GB, 6.5MB/s

Active Disk System

Database Server
- Controller
- UltraSCSI

Server
- Switched Network
- ATM

Controller
- SCSI

Controller
- UltraSCSI

Controller
- SCSI

Digital AlphaServer 500/500

Controller
- UltraSCSI

Digital AXP 3000/400 "Active Disks"
Data Mining & Multimedia

Data Mining - association rules \cite{Agrawal95}
- frequent sets summary counts
- milk & bread $\Rightarrow$ cheese

Database - nearest neighbor search
- $k$ records closest to input record
- with large number of attributes, reduces to scan

Multimedia - edge detection \cite{Smith95}
- detect edges in an image

Multimedia - image registration \cite{Welling97}
- find rotation and translation from reference image
Prototype performance

- factor of 2.5x with Active Disks
- scalable in a more realistic, larger system
Performance with Active Disks

<table>
<thead>
<tr>
<th>application</th>
<th>input</th>
<th>computation (inst/byte)</th>
<th>throughput (MB/s)</th>
<th>memory (KB)</th>
<th>selectivity (factor)</th>
<th>bandwidth (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>k=10</td>
<td>7</td>
<td>28.6</td>
<td>72</td>
<td>80,500</td>
<td>0.4</td>
</tr>
<tr>
<td>Frequent Sets</td>
<td>s=0.25%</td>
<td>16</td>
<td>12.5</td>
<td>620</td>
<td>15,000</td>
<td>0.8</td>
</tr>
<tr>
<td>Edge Detection</td>
<td>t=75</td>
<td>303</td>
<td>0.67</td>
<td>1776</td>
<td>110</td>
<td>6.1</td>
</tr>
<tr>
<td>Image Registration</td>
<td>-</td>
<td>4740</td>
<td>0.04</td>
<td>672</td>
<td>180</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Scalable performance

- crossover at four disks - “technology gap”
- cycles/byte => throughput
- selectivity => network bottleneck
Database Systems

Basic Operations

- select - scan
- project - scan & sort
- join - scan & hash-join

Workload

- TPC-D decision support
  - large data, scale factor of 300 GB uses 520 disks
  - ad-hoc queries
  - high-selectivity, “summary” questions
**Digital AlphaServer 8400**

5/625

12 CPUs using Oracle8

---

**Total System Cost** | TPC-D Power | TPC-D Throughput | Price/Performance
---|---|---|---
$2,649,262 | 2406.2 QppD@ 300GB | 986.1 QthD@ 300GB | $1,720 QphD@ 300GB

**Database Size** | Database Manager | Operating System | Other Software | Availability Date
---|---|---|---|---
300GB | Oracle8 v8.0.4 | Digital UNIX V4.0D | None | May 27, 1998

---

**Query Time in Seconds**

448.9

---

**Components**

<table>
<thead>
<tr>
<th>Components</th>
<th>Qty</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>12</td>
<td>612 Mhz DECchip 21164</td>
</tr>
<tr>
<td>Cache Memory per Processor</td>
<td>4MB</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>2</td>
<td>4 GB</td>
</tr>
<tr>
<td>Disk Controllers</td>
<td>29</td>
<td>PCI</td>
</tr>
<tr>
<td>Disks</td>
<td>521</td>
<td>4.3 GB Disks</td>
</tr>
</tbody>
</table>

---

Database Load Time = 22 hours 50 minutes 17 seconds  
Disk Size/Database Size=7.47  
RAID: No
**Active PostgreSQL Select**

**Experimental setup**
- database is PostgreSQL 6.5
- server is 500 MHz Alpha, 256 MB
- disks are Seagate Cheetahs
- vs. $n$ Active Disks
  - 133 MHz Alpha, 64 MB
  - Digital UNIX 3.2g
- ATM networking vs. Ultra SCSI

**performance results**
- SQL `select` operation (selectivity = 52)
- interconnect limited
- scalable Active Disk performance

![Graph showing throughput vs. number of disks]
Database - Aggregation (Project)

select sum(l_price), sum(l_qty)
from lineitem
group by l_return

Relation S

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_shipdate</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_return</th>
</tr>
</thead>
<tbody>
<tr>
<td>1730</td>
<td>01-25-93</td>
<td>6</td>
<td>11051.6</td>
<td>A</td>
</tr>
<tr>
<td>3713</td>
<td>04-12-96</td>
<td>32</td>
<td>29600.3</td>
<td>R</td>
</tr>
<tr>
<td>7010</td>
<td>10-05-98</td>
<td>23</td>
<td>29356.3</td>
<td>A</td>
</tr>
<tr>
<td>32742</td>
<td>05-05-95</td>
<td>8</td>
<td>9281.9</td>
<td>R</td>
</tr>
<tr>
<td>36070</td>
<td>11-27-98</td>
<td>31</td>
<td>34167.9</td>
<td>R</td>
</tr>
</tbody>
</table>
**Query Plan**

1. **SeqScan**
2. **Sort**
3. **Group**
4. **Aggregate**

**Query Text**

```
from lineitem

group by l_return

sum(l_quantity), sum(l_price),
```
Query Plan

```
SeqScan
  Sort
    Group
      Aggregate
```

Query Text

```
sum(l_quantity), sum(l_price),
group by l_return
from lineitem
```

Modification for Active Disks

```
AggGrpSort
  SeqScan
```

```
sum(l_quantity), sum(l_price),
group by l_return
from lineitem
```
Active PostgreSQL Aggregation

Algorithm

- replacement selection sort
- maintain sorted heap in memory
- combine (aggregate) records when keys match exactly

Benefits

- memory requirements determined by output size
- longer average run length
- easy to make *adaptive*

Disadvantage

- poor memory behavior vs. qsort

performance results

- SQL `sum() ... group by` operation (selectivity = 650)
- cycles/byte = 32, cpu limited
**Database - Join**

```sql
select sum(l_price), sum(l_qty)
from lineitem, part
where p_name like '%green%'
and l_partkey = p_partkey
group by l_return
```

<table>
<thead>
<tr>
<th>l_return</th>
<th>sum_revenue</th>
<th>sum_qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40407.9</td>
<td>29</td>
</tr>
<tr>
<td>R</td>
<td>34167.9</td>
<td>31</td>
</tr>
</tbody>
</table>

**Relation S**

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_partkey</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_return</th>
</tr>
</thead>
<tbody>
<tr>
<td>1730</td>
<td>2593</td>
<td>6</td>
<td>11051.6</td>
<td>A</td>
</tr>
<tr>
<td>3713</td>
<td>0412</td>
<td>32</td>
<td>29600.3</td>
<td>R</td>
</tr>
<tr>
<td>7010</td>
<td>1098</td>
<td>23</td>
<td>29356.3</td>
<td>A</td>
</tr>
</tbody>
</table>

**Relation R**

<table>
<thead>
<tr>
<th>p_partkey</th>
<th>p_name</th>
<th>p_brand</th>
<th>p_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2593</td>
<td>green car</td>
<td>vw</td>
<td>11</td>
</tr>
<tr>
<td>5059</td>
<td>red boat</td>
<td>fast</td>
<td>29</td>
</tr>
<tr>
<td>1098</td>
<td>green tree</td>
<td>pine</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p_partkey</th>
<th>p_name</th>
<th>p_brand</th>
<th>p_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0412</td>
<td>blue sky</td>
<td>clear</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>5692</td>
<td>red river</td>
<td>dirty</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>
Bloom Join

```
select sum(l_price), sum(l_qty)
from lineitem, part
where p_name like '%green%'
and l_partkey = p_partkey
group by l_return
```
Bloom Join

```
<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_partkey</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_return</th>
</tr>
</thead>
<tbody>
<tr>
<td>1730</td>
<td>2593</td>
<td>6</td>
<td>11051.6</td>
<td>A</td>
</tr>
<tr>
<td>7010</td>
<td>1098</td>
<td>23</td>
<td>29356.3</td>
<td>A</td>
</tr>
<tr>
<td>36070</td>
<td>2593</td>
<td>31</td>
<td>34167.9</td>
<td>R</td>
</tr>
</tbody>
</table>
```

```sql
select sum(l_price), sum(l_qty)
from lineitem, part
where p_name like '%green%'
and l_partkey = p_partkey
group by l_return
```
Bloom Join

```
select sum(l_price), sum(l_qty)
from lineitem, part
where p_name like '%green%' and l_partkey = p_partkey
group by l_return
```
### Bloom Join

#### Relation S

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_partkey</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_return</th>
</tr>
</thead>
<tbody>
<tr>
<td>1730</td>
<td>2593</td>
<td>6</td>
<td>11051.6</td>
<td>A</td>
</tr>
<tr>
<td>7010</td>
<td>1098</td>
<td>23</td>
<td>29356.3</td>
<td>A</td>
</tr>
<tr>
<td>36070</td>
<td>2593</td>
<td>31</td>
<td>34167.9</td>
<td>R</td>
</tr>
</tbody>
</table>

#### Relation R

<table>
<thead>
<tr>
<th>p_partkey</th>
<th>p_name</th>
<th>p_brand</th>
<th>p_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2593</td>
<td>green car</td>
<td>vw</td>
<td>11</td>
</tr>
<tr>
<td>5059</td>
<td>red boat</td>
<td>fast</td>
<td>29</td>
</tr>
<tr>
<td>1098</td>
<td>green tree</td>
<td>pine</td>
<td>35</td>
</tr>
</tbody>
</table>

#### Bloom Filter

<table>
<thead>
<tr>
<th>l_return</th>
<th>sum_revenue</th>
<th>sum_qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40407.9</td>
<td>29</td>
</tr>
<tr>
<td>R</td>
<td>34167.9</td>
<td>31</td>
</tr>
</tbody>
</table>

#### SQL Query

```sql
select sum(l_price), sum(l_qty)
from lineitem, part
where p_name like '%green%'
and l_partkey = p_partkey
group by l_return
```
Active PostgreSQL Join

**Algorithm**
- read R to host
- create hash table for R
  - generate Bloom filter
- broadcast filter to all disks
- parallel scan at disks
  - semi-join to host
- final join at host

**Performance results**
- SQL 2-way join operation (selectivity = 8)
- will eventually be network limited
Active PostgreSQL Join II

Experimental setup
- database is PostgreSQL 6.5
- server is 500 MHz Alpha, 256 MB
- disks are Seagate Cheetahs
- vs. \( n \) Active Disks
  - 133 MHz Alpha, 64 MB
  - Digital UNIX 3.2g
- ATM networking vs. Ultra SCSI

performance results
- SQL 5-way join operation
- large serial fraction, Amdahl’s Law kicks in
Model Validation (Database)

Select Q1 (5% Match)

Throughput (MB/s)

Number of Disks

Aggregation Q1 (Group By)

Throughput (MB/s)

Number of Disks

Two-Way Join

Throughput (MB/s)

Number of Disks

Join Q9

Throughput (MB/s)

Number of Disks
Database - Summary

Active PostgreSQL Prototype

<table>
<thead>
<tr>
<th>Query</th>
<th>Bottleneck</th>
<th>Traditional (seconds)</th>
<th>Active Disks (seconds)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>computation</td>
<td>76.0</td>
<td>38.0</td>
<td>100%</td>
</tr>
<tr>
<td>Q5</td>
<td>serial fraction</td>
<td>219.0</td>
<td>186.5</td>
<td>17%</td>
</tr>
<tr>
<td>Q6</td>
<td>interconnect</td>
<td>27.2</td>
<td>17.0</td>
<td>60%</td>
</tr>
<tr>
<td>Q9</td>
<td>serial fraction</td>
<td>95.0</td>
<td>85.4</td>
<td>11%</td>
</tr>
</tbody>
</table>

Measured performance

- four most expensive of the 17 TPC-D queries
- compares eight disk systems
- PostgreSQL 6.5 with Active Disk modifications
Database - Extrapolation

Estimated Speedup on Digital 8400 (TPC-D, May 1998)

<table>
<thead>
<tr>
<th>Query</th>
<th>Bottleneck</th>
<th>Traditional (seconds)</th>
<th>Active Disks (seconds)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>computation</td>
<td>4,357.1</td>
<td>307.7</td>
<td>1,320%</td>
</tr>
<tr>
<td>Q5</td>
<td>serial fraction</td>
<td>1988.2</td>
<td>1,470.8</td>
<td>35%</td>
</tr>
<tr>
<td>Q6</td>
<td>interconnect</td>
<td>63.1</td>
<td>6.1</td>
<td>900%</td>
</tr>
<tr>
<td>Q9</td>
<td>serial fraction</td>
<td>2710.8</td>
<td>2,232.1</td>
<td>22%</td>
</tr>
</tbody>
</table>

Predicted performance

- comparison of Digital 8400 with 520 traditional disks
- vs. the same system with 520 Active Disks
Active Disks

Database - Extrapolation

Estimated Speedup on Digital 8400 (TPC-D, May 1998)

<table>
<thead>
<tr>
<th>Query</th>
<th>Bottleneck</th>
<th>Traditional (seconds)</th>
<th>Active Disks (seconds)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>computation</td>
<td>4,357.1</td>
<td>307.7</td>
<td>1,320%</td>
</tr>
<tr>
<td>Q5</td>
<td>serial fraction</td>
<td>1988.2</td>
<td>1,470.8</td>
<td>35%</td>
</tr>
<tr>
<td>Q6</td>
<td>interconnect</td>
<td>63.1</td>
<td>6.1</td>
<td>900%</td>
</tr>
<tr>
<td>Q9</td>
<td>serial fraction</td>
<td>2710.8</td>
<td>2,232.1</td>
<td>22%</td>
</tr>
<tr>
<td>Other Qs</td>
<td></td>
<td>assume unchanged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>18,619.5</td>
<td>13,517.0</td>
<td>38%</td>
</tr>
</tbody>
</table>

Predicted performance

- comparison of Digital 8400 with 520 traditional disks
- vs. the same system with 520 Active Disks
Database - Extrapolation

Estimated Speedup on Digital 8400 (TPC-D, May 1998)

<table>
<thead>
<tr>
<th>Query</th>
<th>Bottleneck</th>
<th>Traditional (seconds)</th>
<th>Active Disks (seconds)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>computation</td>
<td>4,357.1</td>
<td>307.7</td>
<td>1,320%</td>
</tr>
<tr>
<td>Q5</td>
<td>serial fraction</td>
<td>1,988.2</td>
<td>1,470.8</td>
<td>35%</td>
</tr>
<tr>
<td>Q6</td>
<td>interconnect</td>
<td>63.1</td>
<td>6.1</td>
<td>900%</td>
</tr>
<tr>
<td>Q9</td>
<td>serial fraction</td>
<td>2,710.8</td>
<td>2,232.1</td>
<td>22%</td>
</tr>
<tr>
<td>Other Qs</td>
<td>assume unchanged</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>18,619.5</td>
<td>13,517.0</td>
<td>38%</td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td>$2,649,262</td>
<td>$3,034,045</td>
<td>15%</td>
</tr>
</tbody>
</table>

Predicted performance

- comparison of Digital 8400 with 520 traditional disks
- vs. the same system with 520 Active Disks
- overall cost increase of about 15%
  - assuming an Active Disk costs *twice* a traditional disk
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Additional Functionality

Data Mining for Free

- process sequential workload during “idle” time in OLTP
- allows e.g. data mining on an OLTP system

Action in Today’s Disk Drive

1. seek from A to B wait for rotation
2. read block

Modified Action With “Free” Block Scheduling

1a. seek from A to C
1b. read “free” block at C, seek from C to B
2. wait for rotation
3. read block
• combine background and “free” blocks

Integrated scheduling
• possible only at drives
• combines application-level and disk-level information
• achieves 30% of the drives sequential bandwidth “for free”
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Drive-Specific Functionality

Related Work

Contributions & Future Work
Related Work

Database Machines (CASSM, RAP, Gamma)

- today’s advantages - higher disk bandwidth, parallelism
- general-purpose programmability
- parallel databases (Teradata, Tandem, Oracle, IBM)
- CAFS and SCAFS search accelerator (ICL, Fujitsu)

Parallel Programming

- automatic data parallelism (HPF), task parallelism (Fx)
- parallel I/O (Kotz, IBM, Intel)

Parallel Database Operations

- scan [Su75, Ozkarahan75, DeWitt81, ...]
- sort [Knuth73, Salzberg90, DeWitt91, Bleloch97, ...]
- hash-join [Kitsuregawa83, DeWitt85, ...]
Related Work - “Smart Disks”

Intelligent Disks (Berkeley)
- SMP database functions [Keeton98]
- analytic model, large speedups for join and sort (!)
- different architecture - *everything* is iDisks
- disk layout [Wang98], write optimizations

Programming Model (Santa Barbara/Maryland)
- select, sort, image processing via extended SCSI [Acharya98]
- simulation comparisons among Active Disks, Clusters, SMPs
- focus on network bottlenecks

SmartSTOR (Berkeley/IBM)
- analysis of TPC-D, significant benefits possible (!)
- suggest using one processor for multiple disks
- “simple” functions have limited benefits
Contributions

Exploit technology trends
  • “excess” cycles on individual disk drives
  • large systems => lots of disks => lots of power

Analytic
  • performance model - predicts within 25%
  • algorithms & query optimizer - map to Active Disk functions

Prototype
  • data mining & multimedia
    - 2.5x in prototype, scale to 10x
  • database with TPC-D benchmark
    - 20% to 2.5x in prototype, extrapolate 35% to 15x in larger system
  • changed ~2% of database code, run ~5% of code at drives

Novel functionality
  • data mining for free - close to 30% bandwidth “for free”

Conclusion - lots of potential and realistically attainable
Future Work

Extension of Database Functions

- optimization for index-based scans
- update and small request performance

Programming Model - Application Layers

- explicit programmer-controlled?
- vs. fully adaptive application mobility?
- databases have query optimizers, filesystems don’t
- challenges: identify “structure” and identify “functions”

Masses of Storage, Pervasive Storage

- large volumes of data
- really large scale (1,000s or 10,000s of devices)
- MEMS-devices w/ storage and compute, everything is “active”
Detail Slides
Amdahl’s Law

$\text{serial} = S$

$\text{parallel} = \frac{(1 - p) \cdot S + \frac{p \cdot S}{n}}{S}$

Speedup in a Parallel System

- $p$ is parallel fraction
- $(1 - p)$ serial fraction is not improved
Database - Select

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_shipdate</th>
<th>l_qty</th>
<th>l_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>7010</td>
<td>10-05-98</td>
<td>23</td>
<td>29356.3</td>
</tr>
<tr>
<td>36070</td>
<td>11-27-98</td>
<td>31</td>
<td>34167.9</td>
</tr>
</tbody>
</table>

select * from lineitem
where l_shipdate > '01-01-1998'

relation S

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_shipdate</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_disc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1730</td>
<td>01-25-93</td>
<td>6</td>
<td>11051.6</td>
<td>0.02</td>
</tr>
<tr>
<td>3713</td>
<td>04-12-96</td>
<td>32</td>
<td>29600.3</td>
<td>0.07</td>
</tr>
<tr>
<td>7010</td>
<td>10-05-98</td>
<td>23</td>
<td>29356.3</td>
<td>0.09</td>
</tr>
</tbody>
</table>

...

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_shipdate</th>
<th>l_qty</th>
<th>l_price</th>
<th>l_disc</th>
</tr>
</thead>
<tbody>
<tr>
<td>32742</td>
<td>05-05-95</td>
<td>8</td>
<td>9281.9</td>
<td>0.01</td>
</tr>
<tr>
<td>36070</td>
<td>11-27-98</td>
<td>31</td>
<td>34167.9</td>
<td>0.04</td>
</tr>
</tbody>
</table>
Bloom Join

Use only Bloom filter at disks

- semi-join only, final join at host
- fixed-size bit vectors - memory size O(1)!

<table>
<thead>
<tr>
<th>Query</th>
<th>Join</th>
<th>Size of Bloom filter</th>
<th>Keys</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>128 bits 8 kilobytes 64 kilobytes 1 megabyte ideal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>1.1</td>
<td>0.33 0.33 0.33 0.21</td>
<td>12.4</td>
<td>4.2</td>
</tr>
<tr>
<td>Q5</td>
<td>4.1</td>
<td>0.22 0.22 0.22 0.22</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>Q9</td>
<td>1.1</td>
<td>0.11 0.11 0.11 0.05</td>
<td>4.0</td>
<td>4.7</td>
</tr>
<tr>
<td>Q10</td>
<td>2.1</td>
<td>0.33 0.21 0.21 0.08</td>
<td>21.9</td>
<td>28.6</td>
</tr>
</tbody>
</table>

Memory size required at each disk

- from TPC-D queries at 100 GB scale factor
- using a single hash function for all tables and keys
Outline

Motivation

Computation in Storage

Performance Model

Applications & Prototype

Software Structure

Drive-Specific Functionality

Related Work

Contributions & Future Work
Database Primitives

Scan

• evaluate predicate, return matching records
• low memory requirement

Join

• identify matching records in semijoin
• via direct table lookup
• or Bloom filter, when memory is limited

Aggregate/Sort

• replacement selection with record merging
• memory size proportional to result, not input
• runs of length $2^m$ when used in full mergesort
Execute Node

- TupleDesc
- SeqScan
- ExecScan
- Qual
- HeapTuple
- ExprEval
- Heap
- FuncMgr
- Heap
- File
- Disk
- system catalogs
- query parameters
- table schema
- memory page
- disk page
- data type operators
- adt/datetime
- adt/float
- adt/varchar
- adt/network
- adt/geo_ops
- traditional disk
Active Disk Structure

- **Query Parameters**
- **TupleDesc**
- **SeqScan**
- **ExecScan**
- **Qual**
- **HeapTuple**
- **ExprEval**
- **Heap**
- **FuncMgr**
- **System Catalogs**
- **Table Schema**
- **Memory Page**
- **Disk Page**
- **Active Disk**

Data Types:
- Adt/DateTime
- Adt/Float
- Adt/VarChar
- Adt/Network
- Adt/Geo_Ops
- read background blocks only when queue is empty

- vary multiprogramming level - total number of pending requests
- background forced out at high foreground load
- up to 30% response time impact at low load
Data Mining for Free

- read background blocks only when completely “free”

### OLTP Throughput – 1 disk

- Multiprogramming level (MPL) of OLTP vs. throughput (req/s)

### Mining Throughput

- Multiprogramming level (MPL) of OLTP vs. throughput (KB/s)

### OLTP Response Time

- Multiprogramming level (MPL) of OLTP vs. average response time (ms)

**Free block scheduling**

- opportunistic read
- constant background bandwidth, even at highest loads
- no impact on foreground respond time
Data Mining for Free

- combine background and “free” blocks

Integrated scheduling
- possible only at drives
- combines application-level and disk-level information
- achieves 30% of the drives sequential bandwidth “for free”
Extra Slides
Why Isn’t This Parallel Programming?

It is

- parallel cores
- distributed computation
- serial portion needs to be small

Disks are different

- must protect the data, can’t “just reboot”
- must continue to serve demand requests
- memory/CPU ratios driven by cost, reliability, volume
- come in boxes of ten
- *basic advantage* - compute close to the data

**Opportunistically use this power**

- e.g. data mining possible on an OLTP system
- ok to “waste” the power if it can’t be used
Application Characteristics

Critical properties for Active Disk performance

- cycles/byte => maximum throughput
- memory footprint
- selectivity => network bandwidth

<table>
<thead>
<tr>
<th>application</th>
<th>input</th>
<th>computation (instr/byte)</th>
<th>throughput (MB/s)</th>
<th>memory (KB)</th>
<th>selectivity (factor)</th>
<th>bandwidth (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>m=1%</td>
<td>7</td>
<td>28.6</td>
<td>-</td>
<td>100</td>
<td>290</td>
</tr>
<tr>
<td>Search</td>
<td>k=10</td>
<td>7</td>
<td>28.6</td>
<td>72</td>
<td>80,500</td>
<td>0.4</td>
</tr>
<tr>
<td>Frequent Sets</td>
<td>s=0.25%</td>
<td>16</td>
<td>12.5</td>
<td>620</td>
<td>15,000</td>
<td>0.8</td>
</tr>
<tr>
<td>Edge Detection</td>
<td>t=75</td>
<td>303</td>
<td>0.67</td>
<td>1776</td>
<td>110</td>
<td>6.1</td>
</tr>
<tr>
<td>Image Registration</td>
<td>-</td>
<td>4740*</td>
<td>0.04</td>
<td>672</td>
<td>180</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Select m=20% 7 28.6 - 5 5,700
Frequent Sets s=0.025% 16 12.5 2,000 14,000 0.9
Edge Detection t=20 394 0.51 1750 3 170
Sorts

Local Sort Phase

- replacement selection in Active Disk memory process as data comes off the disk
- build sorted runs of average size $2m$
- can easily adapt to changes in available memory

Local Merge Phase

- perform sub-merges at disks
  less runs to process at host
- also adaptable to changes in memory

Global Merge Phase

- moves all data to the host and back

Optimizations

- duplicate removal, aggregation lower requirements
  memory required only for result, not source relations

Bottleneck is the Network - the Data Must Move Once

- so goal is optimal utilization of links
**Sort Performance**

**Network is the bottleneck**
- Active Disks benefit from reduced interconnect traffic
- using key-only sort improves both systems
- with direct disk to disk transfers, data *never* goes to the host
Database - Joins

Size of R determines Active Disk partitioning

- if $|R| \ll |S|$ (R is the inner, smaller relation)
  - and $|R| < |Active Disk memory|$  
    embarassingly parallel, linear speedup
  - and $|R| < |Server memory|$  
    retain portion of R at each disk, and “assist” Server
- if $|R| \approx |S|$ and $|R| > |Server memory|$  
  process R in parallel, minimize network traffic
- pre-join scan on S and R is always a win  
  reduces interconnect traffic

Assumptions
- non-indexed keys
- not partition keys
- large S (multi-GB)  
  => many disks

Active Disk System

- Controller
- Obj Stor
- Network
- Security

Server

Switched Network

4,096 MB

4 - 64 MB

AlphaServer 8400 TPC-D
- 521 disks, low CPU cost
- network bottlenecked
Join Performance

benefits from reduced interconnect traffic

- determinant is relative size of inner and outer relations
- savings in network transfer
- vs. multiple passes at disks
Database - TPC-D Query 1

**Query Text**

```
select l_returnflag, l_linestatus, 
sum(l_quantity), sum(l_price), 
sum(l_price*(1-l_disc)), sum(l_price*(1-l_disc)*(1+l_tax)), 
avg(l_quantity), avg(l_price), avg(l_disc), count(*)
order by l_returnflag, l_linestatus
```

**Data Reduction**

- Sort: 9 -> 9
- Aggr: 33,935 -> 9
- Group: 33,935 -> 33,935
- Sort: 33,935 -> 33,935
- Qual: 35,189 -> 33,935
- Scan: 126,440 -> 35,189

**Query Plan**

- Scan: 126,440 KB (15,805 pages) on disk
- **Query Result**

```
l_rf | l_ls | sum_qty | sum_base_price | sum_disc_price | sum_charge | avg_qty | price | disc | count
-----|-----|--------|----------------|---------------|-----------|--------|-------|-----|------
A    | F   | 3773034| 5319329289.67 | 5053976845.78 | 5256336547.67 | 25.509 | 35964.01 | 0.049 | 147907
N    | F   | 100245 | 141459686.10  | 134380852.77  | 139710306.87  | 25.625 | 36160.45 | 0.050 | 3912
N    | O   | 7464940| 10518546073.97 | 9992072944.46 | 10392414192.06 | 25.541 | 35990.12 | 0.050 | 292262
R    | F   | 3779140| 5328886172.98  | 5062370635.93 | 5265431221.82  | 25.548 | 36025.46 | 0.050 | 147920
       |     |        |                |               |            | (4 rows) |
```
Database - Data Reduction

Data Reduction for Sequential Scan and Aggregation

<table>
<thead>
<tr>
<th>Query</th>
<th>Input Data (KB)</th>
<th>SeqScan Result (KB)</th>
<th>SeqScan Savings (selectivity)</th>
<th>Aggregate Result (bytes)</th>
<th>Aggregate Savings (selectivity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>126,440</td>
<td>34,687</td>
<td>3.6</td>
<td>240</td>
<td>147,997.9</td>
</tr>
<tr>
<td>Q4</td>
<td>29,272</td>
<td>86</td>
<td>340.4</td>
<td>80</td>
<td>1100.8</td>
</tr>
<tr>
<td>Q6</td>
<td>126,440</td>
<td>177</td>
<td>714.4</td>
<td>8</td>
<td>22,656.0</td>
</tr>
</tbody>
</table>

Input Table

<table>
<thead>
<tr>
<th>l_okey</th>
<th>l_quantity</th>
<th>l_price</th>
<th>l_disc</th>
<th>l_rf</th>
<th>l_is</th>
<th>l_shipdate</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
<th>l_shipmode</th>
<th>l_comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1730</td>
<td>6</td>
<td>11051.58</td>
<td>0.02</td>
<td>N</td>
<td>09-02-1998</td>
<td>10-10-1998</td>
<td>09-13-1998</td>
<td>TRUCK</td>
<td>wSRnnC2</td>
</tr>
<tr>
<td></td>
<td>3713</td>
<td>32</td>
<td>29600.32</td>
<td>0.07</td>
<td>0.03</td>
<td>09-02-1998</td>
<td>06-11-1998</td>
<td>09-28-1998</td>
<td>TRUCK</td>
<td>MQgnCOl</td>
</tr>
<tr>
<td></td>
<td>7010</td>
<td>23</td>
<td>29356.28</td>
<td>0.09</td>
<td>0.06</td>
<td>09-02-1998</td>
<td>08-01-1998</td>
<td>09-14-1998</td>
<td>MAIL</td>
<td>jPNQL2x3i</td>
</tr>
<tr>
<td></td>
<td>19876</td>
<td>4</td>
<td>6867.24</td>
<td>0.09</td>
<td>0.08</td>
<td>09-02-1998</td>
<td>09-06-1998</td>
<td>09-29-1998</td>
<td>AIR</td>
<td>3nRkN4</td>
</tr>
<tr>
<td></td>
<td>24839</td>
<td>8</td>
<td>12845.52</td>
<td>0.05</td>
<td>0.02</td>
<td>09-02-1998</td>
<td>10-14-1998</td>
<td>09-06-1998</td>
<td>REG AIR</td>
<td>jlw61g3</td>
</tr>
<tr>
<td></td>
<td>25217</td>
<td>10</td>
<td>18289.1</td>
<td>0.05</td>
<td>0.07</td>
<td>09-02-1998</td>
<td>08-12-1998</td>
<td>09-26-1998</td>
<td>TRUCK</td>
<td>SQ7xS5</td>
</tr>
<tr>
<td></td>
<td>29348</td>
<td>29</td>
<td>41688.08</td>
<td>0.05</td>
<td>0.02</td>
<td>09-02-1998</td>
<td>07-04-1998</td>
<td>09-18-1998</td>
<td>FOB</td>
<td>C0Nxhr2M</td>
</tr>
<tr>
<td></td>
<td>32742</td>
<td>8</td>
<td>9281.92</td>
<td>0.01</td>
<td>0.03</td>
<td>09-02-1998</td>
<td>07-17-1998</td>
<td>09-19-1998</td>
<td>FOB</td>
<td>N3MO1C</td>
</tr>
<tr>
<td></td>
<td>36070</td>
<td>31</td>
<td>34167.89</td>
<td>0.04</td>
<td>N</td>
<td>09-02-1998</td>
<td>07-11-1998</td>
<td>09-21-1998</td>
<td>REG AIR</td>
<td>k10wyR</td>
</tr>
</tbody>
</table>

(600752 rows)
select
sum(l_price*l_disc)
where l_shipdate >= '1994-01-01'
and l_shipdate < '1995-01-01'
and l_disc between 0.05 and 0.07
and l_quantity < 24
from lineitem

126,440 KB (15,805 pages) on disk

<table>
<thead>
<tr>
<th>Data Reduction</th>
<th>Query Plan</th>
<th>Query Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggr</td>
<td>SeqScan</td>
<td>select</td>
</tr>
<tr>
<td>43 -&gt; 1</td>
<td></td>
<td>sum(l_price*l_disc)</td>
</tr>
<tr>
<td>Qual</td>
<td>Aggr</td>
<td>where l_shipdate &gt;= '1994-01-01'</td>
</tr>
<tr>
<td>9,383 -&gt; 43</td>
<td></td>
<td>and l_shipdate &lt; '1995-01-01'</td>
</tr>
<tr>
<td>Scan</td>
<td></td>
<td>and l_disc between 0.05 and 0.07</td>
</tr>
<tr>
<td>126,440 -&gt; 9,383</td>
<td></td>
<td>and l_quantity &lt; 24</td>
</tr>
</tbody>
</table>

Query Result

```
revenue
---------
11450588.04
(1 row)
```
How to split operations between host and drives?

**Answer:** Use existing query optimizer

- operation costs
- per-table and per-attribute statistics
- ok if they are slightly out-of-date, only an estimate

---

### Query Input Data

<table>
<thead>
<tr>
<th>Query</th>
<th>Input Data (KB)</th>
<th>Scan Result (KB)</th>
<th>Optimizer Estimate (KB)</th>
<th>Qualifier Result (KB)</th>
<th>Optimizer Estimate (KB)</th>
<th>Aggregate Result (bytes)</th>
<th>Optimizer Estimate (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>126,440</td>
<td>35,189</td>
<td>35,189</td>
<td>34,687</td>
<td>33,935</td>
<td>240</td>
<td>9,180</td>
</tr>
<tr>
<td>Q4</td>
<td>29,272</td>
<td>2,343</td>
<td>2,343</td>
<td>86</td>
<td>141</td>
<td>80</td>
<td>64</td>
</tr>
<tr>
<td>Q6</td>
<td>126,440</td>
<td>9,383</td>
<td>9,383</td>
<td>177</td>
<td>43</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

---

**Move ops to drives if there are sufficient resources**

- if selectivity and parallelism overcome slower CPU

**Be prepared to revert to host as two-stage algorithm**

- consider the disk as “pre-filtering”
- still offloads significant host CPU and interconnect
### Database - Optimizer Statistics

#### Statistics

<table>
<thead>
<tr>
<th>starel1d</th>
<th>staattnum</th>
<th>staop</th>
<th>stalokey</th>
<th>stahikey</th>
</tr>
</thead>
<tbody>
<tr>
<td>18663</td>
<td>1</td>
<td>66</td>
<td>1</td>
<td>600000</td>
</tr>
<tr>
<td>18663</td>
<td>2</td>
<td>66</td>
<td>1</td>
<td>20000</td>
</tr>
<tr>
<td>18663</td>
<td>3</td>
<td>66</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>18663</td>
<td>4</td>
<td>66</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>18663</td>
<td>5</td>
<td>295</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>18663</td>
<td>6</td>
<td>295</td>
<td>901</td>
<td>95949.5</td>
</tr>
<tr>
<td>18663</td>
<td>7</td>
<td>295</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>18663</td>
<td>8</td>
<td>295</td>
<td>0</td>
<td>0.08</td>
</tr>
<tr>
<td>18663</td>
<td>9</td>
<td>1049</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>18663</td>
<td>10</td>
<td>1049</td>
<td>F</td>
<td>O</td>
</tr>
<tr>
<td>18663</td>
<td>11</td>
<td>1087</td>
<td>01-02-1992</td>
<td>12-01-1998</td>
</tr>
<tr>
<td>18663</td>
<td>12</td>
<td>1087</td>
<td>01-31-1992</td>
<td>10-31-1998</td>
</tr>
<tr>
<td>18663</td>
<td>13</td>
<td>1087</td>
<td>01-08-1992</td>
<td>12-30-1998</td>
</tr>
<tr>
<td>18663</td>
<td>14</td>
<td>1049</td>
<td>COLLECT COD</td>
<td>TAKE BACK RETURN</td>
</tr>
<tr>
<td>18663</td>
<td>15</td>
<td>1049</td>
<td>AIR</td>
<td>TRUCK</td>
</tr>
<tr>
<td>18663</td>
<td>16</td>
<td>1049</td>
<td>0B6wmAw2Pg</td>
<td>zzyRPS40ABMRSzmPyCNzA6</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>attrelid</th>
<th>attname</th>
<th>attypid</th>
<th>attdisbursion</th>
<th>attlen</th>
<th>attnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>18663</td>
<td>l_orderkey</td>
<td>23</td>
<td>2.33122e-06</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>18663</td>
<td>l_partkey</td>
<td>23</td>
<td>1.06588e-05</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>18663</td>
<td>l_suppkey</td>
<td>23</td>
<td>0.000213367</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>18663</td>
<td>l_linenumber</td>
<td>23</td>
<td>0.0998572</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>18663</td>
<td>l_quantity</td>
<td>701</td>
<td>0.00434997</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>18663</td>
<td>l_extendedprice</td>
<td>701</td>
<td>2.66427e-06</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>18663</td>
<td>l_discount</td>
<td>701</td>
<td>0.0247805</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>18663</td>
<td>l_tax</td>
<td>701</td>
<td>0.0321099</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>18663</td>
<td>l_returnflag</td>
<td>1042</td>
<td>0.307469</td>
<td>-1</td>
<td>9</td>
</tr>
<tr>
<td>18663</td>
<td>l_linenstatus</td>
<td>1042</td>
<td>0.300911</td>
<td>-1</td>
<td>10</td>
</tr>
<tr>
<td>18663</td>
<td>l_shipdate</td>
<td>1082</td>
<td>8.94076e-05</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>18663</td>
<td>l_commitdate</td>
<td>1082</td>
<td>8.33926e-05</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>18663</td>
<td>l_receiptdate</td>
<td>1082</td>
<td>8.90733e-05</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>18663</td>
<td>l_shipinstruct</td>
<td>1042</td>
<td>0.100238</td>
<td>-1</td>
<td>14</td>
</tr>
<tr>
<td>18663</td>
<td>l_shipmode</td>
<td>1042</td>
<td>0.0451101</td>
<td>-1</td>
<td>15</td>
</tr>
<tr>
<td>18663</td>
<td>l_comment</td>
<td>1042</td>
<td>0</td>
<td>-1</td>
<td>16</td>
</tr>
</tbody>
</table>

(61 rows)
## Active PostgreSQL - Code Changes

<table>
<thead>
<tr>
<th>Module</th>
<th>Original</th>
<th>Modified Host (New &amp; Changed)</th>
<th>Active Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Files</td>
<td>Code</td>
<td>Files</td>
</tr>
<tr>
<td>access</td>
<td>72</td>
<td>26,385</td>
<td>-</td>
</tr>
<tr>
<td>bootstrap</td>
<td>2</td>
<td>1,259</td>
<td>-</td>
</tr>
<tr>
<td>catalog</td>
<td>43</td>
<td>13,584</td>
<td>-</td>
</tr>
<tr>
<td>commands</td>
<td>34</td>
<td>11,635</td>
<td>-</td>
</tr>
<tr>
<td>executor</td>
<td>49</td>
<td>17,401</td>
<td>9</td>
</tr>
<tr>
<td>parser</td>
<td>31</td>
<td>9,477</td>
<td>-</td>
</tr>
<tr>
<td>lib</td>
<td>35</td>
<td>7,794</td>
<td>-</td>
</tr>
<tr>
<td>nodes</td>
<td>24</td>
<td>13,092</td>
<td>-</td>
</tr>
<tr>
<td>optimizer</td>
<td>72</td>
<td>19,187</td>
<td>-</td>
</tr>
<tr>
<td>port</td>
<td>5</td>
<td>514</td>
<td>-</td>
</tr>
<tr>
<td>regex</td>
<td>12</td>
<td>4,665</td>
<td>-</td>
</tr>
<tr>
<td>rewrite</td>
<td>13</td>
<td>5,462</td>
<td>-</td>
</tr>
<tr>
<td>storage</td>
<td>50</td>
<td>17,088</td>
<td>1</td>
</tr>
<tr>
<td>tcpop</td>
<td>11</td>
<td>4,054</td>
<td>-</td>
</tr>
<tr>
<td>utils/adt</td>
<td>40</td>
<td>31,526</td>
<td>-</td>
</tr>
<tr>
<td>utils/fmgr</td>
<td>4</td>
<td>2,417</td>
<td>-</td>
</tr>
<tr>
<td>utils</td>
<td>81</td>
<td>19,908</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>578</td>
<td>205,448</td>
<td>10</td>
</tr>
</tbody>
</table>

New 1,257
Code Specialization

<table>
<thead>
<tr>
<th>query</th>
<th>type</th>
<th>computation (instr/byte)</th>
<th>throughput (MB/s)</th>
<th>memory (KB)</th>
<th>selectivity (factor)</th>
<th>instructions (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>aggregation</td>
<td>1.82</td>
<td>73.1</td>
<td>488</td>
<td>816</td>
<td>9.1/4.7</td>
</tr>
<tr>
<td>Q13</td>
<td>hash-join</td>
<td>0.15</td>
<td>886.7</td>
<td>576</td>
<td>967,000</td>
<td>14.3/10.5</td>
</tr>
</tbody>
</table>

Optimized Implementation

- direct C code, single query only, raw binary files
- 133 MHz Alpha 3000/400, Digital UNIX 3.2

<table>
<thead>
<tr>
<th>operation</th>
<th>computation (cycles/byte)</th>
<th>throughput (MB/s)</th>
<th>selectivity (factor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>28</td>
<td>17.8</td>
<td>4.00</td>
</tr>
<tr>
<td>Qualification</td>
<td>29</td>
<td>17.2</td>
<td>1.05</td>
</tr>
<tr>
<td>Sort/Group</td>
<td>71</td>
<td>7.0</td>
<td>1.00</td>
</tr>
<tr>
<td>Sort/Aggregate</td>
<td>196</td>
<td>2.5</td>
<td>3,770.00</td>
</tr>
</tbody>
</table>

Database System

- database manager database is PostgreSQL 6.4.2
- much higher cycles/byte than direct C implementation
  - parses general SQL statements
  - handles arbitrary tuple formats
History - SCAFS

SCAFS (Son of Content-Addressable File Store)
- processing unit in a 3.5” form factor, fit into a drive shelf
- communication via SCSI commands

Goals
- invisible to the application layer (i.e. hidden under SQL)
- established as an industry-standard for high volume market

Benefits
- 40% to 3x throughput improvement in a mixed workload
- 20% to 20x improvement in response time
- 2x to 20x for a “pure” decision support workload
- up to 100x improvement in response time
Lessons from CAFS [Anderson98]

Why did CAFS not become wildly popular?

- “synchronization was a big problem”
  \textit{Answer} - Yes. Major concern for OLTP, less for “mining”.

- “dynamic switching between applications is a problem”
  \textit{Answer} - Yes. But operating systems know how to do this.

- “not the most economical way to add CPU power”
  \textit{Answer} - but it \textit{is} the best bandwidth/capacity/compute combo and you can still add CPU if that helps (and if you can keep it fed)

- “CPU is a more flexible resource”, disk processor wasted when not in use
  \textit{Answer} - you’re already wasting it today, silicon is everywhere

- “memory size is actually a bigger problem”
  \textit{Answer} - use adaptive algorithms, apps have “sweet spots”

- “needed higher volume, lower cost function”
  \textit{Answer} - this is exactly what the drive vendors can provide no specialized, database-specific hardware necessary

- “could not get it to fit into the database world”
  \textit{Answer} - proof of concept, community willing to listen
Yesterday’s Server-Attached Disks

Store-and-forward data copy through server machine

File/Database Server
Local Area Network

Controller
SCSI

Controller
SCSI

Controller
SCSI

Controller
SCSI

Separate storage and client networks
- storage moving to packetized FC
- clients moving to scalable switches
Network-Attached Secure Disks

Eliminate server bottleneck w/ network-attached
- server scaling [SIGMETRICS ‘97]
- object interface, filesystems [CMU-CS ‘98]
- cost-effective, high bandwidth [ASPLOS ‘98]

Combined storage and client networks
- single, switched infrastructure
- delivers max. bandwidth to clients
- drives must handle security
# TPC-D Benchmark

Consists of *high selectivity, ad-hoc queries*

<table>
<thead>
<tr>
<th>query</th>
<th>entire query</th>
<th></th>
<th>scan only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input (MB)</td>
<td>result (KB)</td>
<td>input (MB)</td>
</tr>
<tr>
<td>Q1</td>
<td>672</td>
<td>0.2</td>
<td>672</td>
</tr>
<tr>
<td>Q5</td>
<td>857</td>
<td>0.09</td>
<td>672</td>
</tr>
<tr>
<td>Q7</td>
<td>857</td>
<td>0.02</td>
<td>672</td>
</tr>
<tr>
<td>Q9</td>
<td>976</td>
<td>6.5</td>
<td>672</td>
</tr>
<tr>
<td>Q11</td>
<td>117</td>
<td>0.3</td>
<td>115</td>
</tr>
</tbody>
</table>

Scale Factor = 1 GB

**Simple filtering on input**
- factors of 3x and more savings in load on interconnect

**Entire queries (including aggregation and joins)**
- factors of 100,000 and higher savings
Implementation Issues

Partitioning

- combining disk code with “traditional” code

Mobility

- code must run on disks and/or host
- Java (!) (?)
  + popular, tools (coming soon), strong typing
  - somewhat different emphasis what to optimize for
- more “static” extensions

Interfaces

- capability system of NASD as a base
- additional inquiry functions for scheduling
- additional power (via capabilities) for storage mgmt
Value-Added Storage

Variety of value-added storage devices

<table>
<thead>
<tr>
<th>System</th>
<th>Function</th>
<th>Cost</th>
<th>Premium</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate Cheetah 18LP LVD</td>
<td>disk only</td>
<td>$900</td>
<td>-</td>
<td>18 GB, lvd, 10,000 rpm</td>
</tr>
<tr>
<td>Seagate Cheetah 18LP FC</td>
<td>disk only</td>
<td>$942</td>
<td>5%</td>
<td>FC</td>
</tr>
<tr>
<td>Dell 200S PowerVault</td>
<td>drive shelves &amp; cabinet</td>
<td>$10,645</td>
<td>48%</td>
<td>8 lvd disks</td>
</tr>
<tr>
<td>Dell 650F PowerVault</td>
<td>dual RAID controllers</td>
<td>$32,005</td>
<td>240%</td>
<td>10 disks, full FC</td>
</tr>
<tr>
<td>Dell 720N PowerVault</td>
<td>CIFS, NFS, Filer</td>
<td>$52,495</td>
<td>248%</td>
<td>16 disks, ether, 256/8 cache</td>
</tr>
<tr>
<td>EMC Symmetrix 3330-18</td>
<td>RAID, management</td>
<td>$160,000</td>
<td>962%</td>
<td>16 disks, 2 GB cache</td>
</tr>
</tbody>
</table>

Price premium

- cabinet cost is significant
- network-attached storage is as costly as RAID
- “management” gets the biggest margin
Network “Appliances” Can Win Today

**Dell PowerEdge & PowerVault System**

Dell PowerVault 650F  
$40,354 \times 12 = 484,248$
- 512 MB cache, dual link controllers, additional 630F cabinet,
- 20 x 9 GB FC disks, software support, installation

Dell PowerEdge 6350  
$11,512 \times 12 = 138,144$
- 500 MHz PIII, 512 MB RAM, 27 GB disk

3Com SuperStack II 3800 Switch  
$7,041$
- 10/100 Ethernet, Layer 3, 24-port

Rack Space for all that  
$20,710$

**NASRaQ System**

Cobalt NASRaQ  
$1,500 \times 240 = 360,000$
- 250 MHz RISC, 32 MB RAM, 2 x 10 GB disks

Extra Memory (to 128 MB each)  
$183 \times 360 = 65,880$

3Com SuperStack II 3800 Switch  
$7,041 \times 11 = 77,451$
- 240/24 = 10 + 1 to connect those 10

Dell PowerEdge 6350 Front-End  
$11,512$

Rack Space (estimate 4x as much as the Dells)  
$82,840$

Installation & Misc  
$50,000$

<table>
<thead>
<tr>
<th></th>
<th>Dell</th>
<th>Cobalt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>2.1 TB</td>
<td>4.7 TB</td>
</tr>
<tr>
<td><strong>Spindles</strong></td>
<td>240</td>
<td>480</td>
</tr>
<tr>
<td><strong>Compute</strong></td>
<td>6 GHz</td>
<td>60 GHz</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>12.3 GB</td>
<td>30.7 GB</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>23,122 W</td>
<td>12,098 W</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>$650,143</td>
<td>$647,683</td>
</tr>
</tbody>
</table>