Why trouble humans?
They do not care!

Toni Cortes
Storage-system Research Group
Barcelona Supercomputing Center
I know “exascaling” is the hot topic today...
  – Especially in a conference about parallelism
... but you have probably heard too much about it already!
I propose to talk about parallelism (maybe “exascaling” too)
... but form a different point of view ...

... the people!
While using a “large” system
- How many times have you had to
  - Adapt your code or “behavior” to what the system expects?
    - Did you know why?
    - Did the motivation behind make any sense to you?
  - Tune parameters you do not understand?
- And ... you are the experts!!!

But, who is to blame?
- When solving a research challenge
  - How many new parameters have you added?
  - What is the increased complexity for users?

... But things may be changing!
Parallel problems are hard …
… Exaflop systems are even harder …
… Should we put the pressure on users?
Happy users: best adopters of new technology
  – Programmers
  – System administrators
  – Scientists
  – Untrained users
  – …

No matter how hard your problem is, do not put the load on the user!
Did we ever think that a Cray2 (1st in Top 500 in 1985) ... 
... or any system in the Top500 till 1994 ... 
... could be as easy to use as an iPad in 2011?

Why not adding usability as a top requirement?
Three examples for today

- **Do not put too many files in a single directory!**
  - But, I want to store a file per process!
    - And I have millions!
  - *Humans do not care or understand!*

- **The best disk scheduler depends on the load, device, ...**
  - But, I need to configure one!
    - I always want the best I/O performance!
  - *Humans do not know!*

- **You should pick the fastest replica**
  - But how do I relate distance with latency?
    - And bandwidth?
  - *Humans do not know, understand, or care!*
This is not a storage talk
  - Though all examples are about storage

It is about designing usable parallel/distributed systems
  - Do not forget: humans have to use them

The lessons to take home can be applied to any field

Should!
Example #1: Millions of files in a directory
If you create too many files in a directory ...

The solution system administrators give:
- Distribute your files among several directories

Does this make any sense?
Background

- Current system already do many things well
  - It makes no sense to build a new system (at least for us)

Objective:

- We want to solve the name space scalability
  - Directory locking (especially if many clients)
  - Internal structures
  - Basically → directories with few files work!
- Without breaking what is already working

Idea

- Decouple the view users have from the system implementation
  - User view: many files in a single directory
  - System implementation: directories as the system likes them
Proposed translation function
- Hash (client, Parent directory, Process, Random “bits”)
  - Guarantees that a physical directory never has more than 512 files!

Example
- User file
  - /dir/fileA
- Actual file
  - /0tr5/0j7a6/65fh
Architecture

Namespace server

Name space database

Handles:
• Name spaces
• Some metadata
• Virtual directory locking

Redirects to:
• Namespace server
• Storage nodes

Caches:
• Placement (w/ leases)

Storage nodes

Handles:
• Physical directory locking
• Regular data accesses
• Some metadata

Storage nodes

(dir/file)

/0er5/T539/66gg
Results
Lessons learnt

- Question assumptions regularly
  - Who cared about metadata 20 years ago?
- Decoupling user view from implementation is good
  - Traditional mechanisms may not be the best

Future
- Lessons learnt should be included in real systems
Example #2: Adaptable disk scheduler
Let’s only compare CFQ and deadline

- Linux kernel read 32 processes works 25% better with deadline
  ... but IO-R 32 processes works 60% better with CFQ
  
  Depends on the application!

- Linux kernel read
  - When running with 8 process, CFQ is 25% better
  - When running with 32 processes, deadline is 15% better
  
  Depends on the number of processes!

- TAC benchmark with 4 processes
  - Using a Seagate ST3250310NS, CFQ is 20% better
  - Using a Seagate ST3500630AS, Deadline is 20% better
  
  Depends on the disk used!

How can I know?
Objective

- Background
  - Current systems have several disk schedulers
    - Their performance depends on the load, devices, ...
    - Very difficult to choose (and to tune their parameters)
  - New architectures have many cores
    - They will not be always used
    - Cores will be like memory
      - Who uses all memory?

- Objective:
  - A mechanism that, on-line, selects the best scheduler
    - ... and its parameters

- Idea
  - Use available cores to perform time-consuming analysis
Cluster of requests

- Assumption: similar patterns need same disk scheduler

- From access patterns to clusters
  - Capture the trace of I/O requests
  - Reduce the trace (we want to support 1 Million ops per second)
    - Divide the trace in 1000 buckets
      - Select the max or min sector jump randomly
    - Include IOPs
  - Cluster reduced patterns using FastDTW

- Using clusters
  - Learn the best scheduler per cluster
    - Current version: by trial and error
  - Learn sequences of clusters
  - When a cluster is detected, change to best scheduler for next one
These operations *are* quite time consuming
- Reducing traces
  - Needs to be done fast
- FastDWT
  - Quite time consuming, but not in critical path

These operations *will be* quite time consuming
- Use simulations to predict the behavior of an scheduler
  - We could try many more parameters
  - In our roadmap for the near future
Results using TCP-E

Amazon/ebay load
4000 clients, 80 trade days, Scale factor 500
MySQL 5.1.41

Improvement over NOOP

<table>
<thead>
<tr>
<th>Disk scheduler</th>
<th>DL</th>
<th>AS</th>
<th>CFQ</th>
<th>DYN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20%</td>
<td>15%</td>
<td>15%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Disk scheduler
Lessons learnt

- We will have plenty of cores in future systems
  - Why not use them to improve system software?
    - No all of them will used anyway!

- Many things were not possible in the past
  - Can we revisit them?
Example #3: Automatic location of replicas
The problem

Oslo

Barcelona

Varsaw

Patra

2145Km 73ms

1865Km 64ms

1705Km 90ms

How can a user guess?
Objective

Background

- Vivaldi
  - Mechanisms to map latency distances in an Euclidean space
  - Dynamically adapts to changes in latency
  - Not the only mechanism

Objective

- Apply this idea to XtreemFS/XtreemOS
  - When using a file
    - Select the closest replica to a client (latency wise)
  - When launching a job
    - Select resources close to replicas (latency wise)

Idea

- Map clients and OSDs in a 2D space
  - Use Euclidean distance
Computes new coordinates using:
- Coordinates
- RTT

Periodically:
Selects an OSD randomly

Computes new coordinates using:
• Coordinates
• RTT

Accesses closest replica

Open (file)
Results

<table>
<thead>
<tr>
<th>Latency in milliseconds</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Best
- 1-2 best
- 1-3 best
Lessons learnt

- Very simple ideas can work fairly well
  - How much accuracy can you lose to make it ...
    - Usable?
    - Scalable?
General conclusions

**Wish #1**

Conclusion #1
- I hope I cannot reuse these slides in 10 years

Other conclusions
- Question assumptions regularly
- Traditional mechanisms may not be the best
- Use available computing power to revisit old ideas
- Approximate solutions are good in many cases
  - And they are usable!
  - And they scale!
Thanks to...

- Ernest Artiaga (BSC)
- Jacobo Giralt (BSC)
- Juan González (BSC)
- Pilar González (U. Murcia)
- Ramón Nou (BSC)
- Jesús Malo (now at Quiterian)
- Joanthan Martí (BSC)
- Juan Piernas (U. Murcia)

And the storage-system research group at BSC in general