Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching

Eddy Caron, Frédéric Desprez, **Adrian Mureșan**

Ecole Normale Supérieure de Lyon, France

2^{nd} December 2010
CloudCom2010 – 2^{nd} IEEE International Conference on Cloud Computing Technology and science

Outline

A typical Cloud client

- Provides a form of Web Service
- Deploys a user-interactive application

A typical Cloud client

- Provides a form of Web Service
- Deploys a user-interactive application

Cloud client goals

- Take advantage of the Cloud's flexibility
- Have a higher resource usage efficiency
- Scale his application according to need
- *Reduce expenses*

The situation

+ IaaS Cloud providers have APIs for platform manipulation
− Virtual resources have a setup time
    - $\approx$ 1 min 22 sec for an EC2 m1.small instance

Setup time The total time it takes for the virtual resource to be usable since the request was issued.

Outline

Our starting point

Jonathan Kupferman, Jeff Silverman, Patricio Jara, Jeff Browne. UCSB. "Scaling Into The Cloud"

- Analyzed three algorithms for dynamic Cloud Client resource scaling
- Proposed a new scoring metric for auto-scaling algorithms

Conclusions

- Dynamic provisioning provides large improvements over static allocation
- Predictive approaches tend to respond more rapidly to sharp changes

What can currently be found in practice as auto-scaling algorithms

Reactive: scaling decisions are made as a function of the current platform's state

Predictive: considers past platform states as a discrete function and constructs a mathematical model to extrapolate

Reactive approaches

Examples:

- The Rightscale algorithm
- Elasticity rules based scaling

Pros / cons:

+ Offered by most Cloud providers
+ Simple to setup and use
− Immune to platform repetitive behavior
− Virtual resource setup time is still a problem

Predictive approaches

Examples:

- Regressive approaches
- Moving averages
- Neural network
- ...
- [our pattern-based approach]

Pros / cons:

+ Can compensate for virtual resource setup time
+ Periodic repetitive behavior
+ More insight than reactive approaches
− Non-periodic repetitive behavior *
− Prediction accuracy can be a problem

Outline

What we propose

- A pattern-based predictive approach
- Compensate for the virtual resource setup time
- Predict future Cloud client application load by identifying similar past usage patterns

What we propose

- A pattern-based predictive approach
- Compensate for the virtual resource setup time
- Predict future Cloud client application load by identifying similar past usage patterns

Self-similarity in web traffic

- Technically documented
- Macro level: yearly similarities
- Micro level: serving a file multiple times has a similar load pattern
- **Implication:** repetitive, non-periodic behavior

Our methodology

- Consider a measure of Cloud client platform usage (CPU count, total RAM used, etc.)
- Consider a history of the Cloud client's platform usage of size n
- Consider the last m instances of the measure – the last usage pattern
- Identify p similar usage patterns in the history
- Do a weighted interpolation on the p similar patterns

The resulting values give us an insight into future platform usage.

Our methodology

- Consider a measure of Cloud client platform usage (CPU count, total RAM used, etc.)
- Consider a history of the Cloud client's platform usage of size n
- Consider the last m instances of the measure – the last usage pattern
- Identify p similar usage patterns in the history
- Do a weighted interpolation on the p similar patterns

The resulting values give us an insight into future platform usage.

Side-note

This approach can be used for predicting any measure that has a pattern-like repetitive behavior.

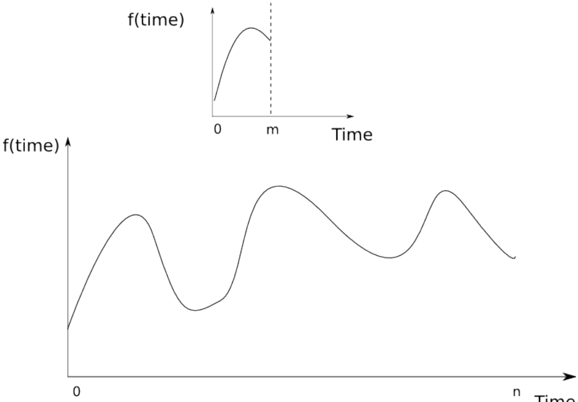

Figure: Mock example #1: historic data; pattern

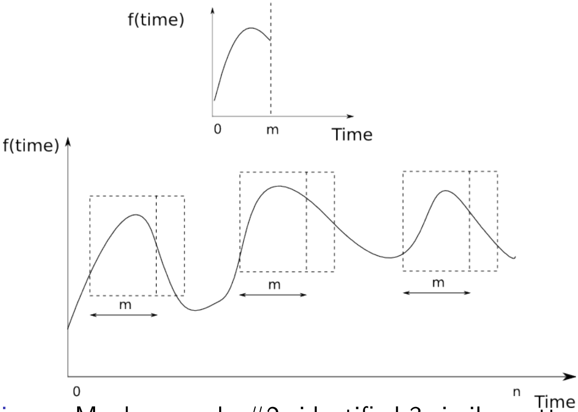

Figure: Mock example #2: identified 3 similar patterns

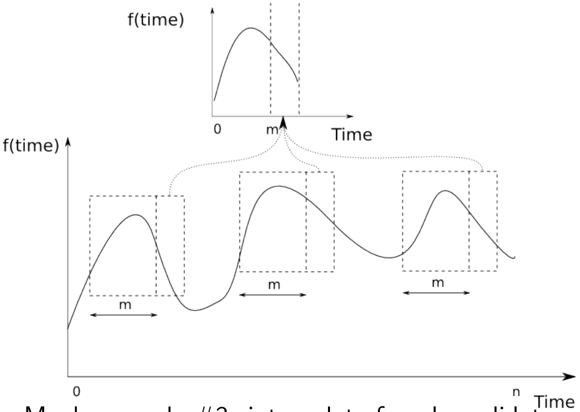

Figure: Mock example #3: interpolate found candidate patterns

Knuth-Morris-Pratt string matching algorithm

+ Used for identifying a substring of length m inside a string of length n
+ Fast running time: $\theta(m+n)$
+ Can be trivially convertible to parallel code if we consider $n >> m$

– Good for exact matches

Knuth-Morris-Pratt string matching algorithm

+ Used for identifying a substring of length m inside a string of length n
+ Fast running time: $\theta(m + n)$
+ Can be trivially convertible to parallel code if we consider $n >> m$
− Good for exact matches

Knuth-Morris-Pratt algorithm adaptation

- Find approximate matches
- Changed running time to $O(m \times n)$ in the worst case

Outline

Data sources

- Animoto Cloud application
- Large Hadron Collider Compute Grid (LCG)
- Nordugrid
- SHARCNET

Data sources

- Animoto Cloud application
- Large Hadron Collider Compute Grid (LCG)
- Nordugrid
- SHARCNET

Call for Cloud Traces

- Cloud traces are needed
- Cloud traces are difficult to obtain
- Open archive would be useful to the community

Experimental methodology

- Used time slices of 100 seconds
- Considered the total number of used CPUs per time slice
- Pattern length of 100 time slices ($\approx$ 2.7 hours)
- Predicted 1 time slice ($\approx$ 1 minute 30 seconds)
- Used traces from one platform to predict
  - its own usage (self-prediction)
  - usage of another platform
- Measured
  - Prediction error
  - Score by using a metric proposed Kupferman et al. (UCSB)

UCSB scoring metric

$$\frac{(A_{log})^{\alpha}}{C} - \frac{\gamma C}{A_{log}} + \beta \tag{1}$$

$A = \frac{\#serviced\_requests}{\#of\_requests}$ represents the availability of the platform

$A_{log} = -log(1 + \delta_a - A), \delta_a < 1$ represents the availability in logarithmic scale

$C = \frac{\#CPU}{hours \times 0.10}$ represents the cost

α, β, γ, δ_a have been chosen through experimentation

Metric	A w/ A	A w/ L	L w/ L	L w/ N	N w/ L	S w/ N
Min err (%)	0.0	0.0	0.0	0.0	0.0	0.0
Max err (%)	100	856.87	53.4	100.0	1146.00	528.03
Med err (%)	2.69	4.08	1.0	1.2	1.74	0.9
Avg err (%)	5.42	7.4	1.749	7.32	35.38	375.65
UCSB	-1.39	-15.95	10.66	3.43	30.64	-3.23
Time (ms)	186	27	41	514	162	528

Table: Prediction results. **A w/ B** – predicting A's usage by using B as historic data. Possible platforms are: Animoto, LCG, NorduGrid and SHARCNET

Pattern length	**Data length**			
	100.0%	50.0%	25.0%	12.5%
1000	5.3%	9.5%	19.7%	100%
500	3.7%	6.0%	8.6%	18.7%
100	1.0%	1.2%	1.3%	2.0%
50	0.6%	0.5%	0.9%	1.3%
25	0.3%	0.3%	0.4%	0.5%
12	0.2%	0.2%	0.2%	0.3%
2	98%	100%	100%	82%

Table: The prediction error obtained for various lengths of historic data and pattern lengths for the LCG platform.

Outline

Conclusions

- The current work presents a resource usage prediction approach based on pattern matching
- Results are good when presented with historic data that is relevant to the current domain
- Results can be improved by
  - increasing historic data size
  - determining the appropriate pattern length
- There is a need for freely-available Cloud platform traces

Future work

- Integrate our proposed approach into a grid and Cloud middleware - DIET http://graal.ens-lyon.fr/DIET/

?