60017 PERFORMANCE ENGINEERING

User Behaviour Modelling

Last lecture

- Benchmarking a distributed application
  - Case study: SPECjbb2015
- Load testing a distributed application

This lecture

- Workloads in computer systems
  - Log files
  - User behavior models

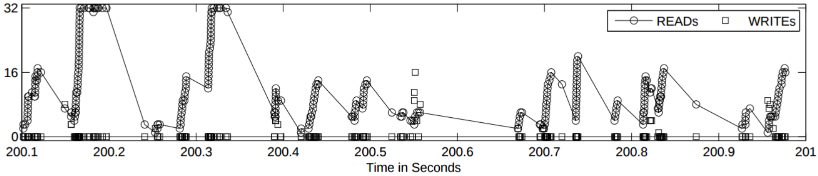
Log files and traces

- Workloads are recorded in log files. Log files record events at operating system or application level and their timestamps.
- Many tools automate log collection, filtering and analysis.
  - *e.g., Elasticsearch, Logstash, Kibana, ...*
- In distributed applications multiple log files exist and need to be combined with appropriate filters into workload traces.
  - *e.g., HTTP log files, log4j files, Linux /var/log files, ...*
- A workload trace organizes events that pertain to arrival and service of requests in a time series.

Log files and traces

- Traces can be used by managers and administrators to:
  - understand how the system is used by customers
  - simulate the system to answer what-if questions
  - replay a sequence of requests using a load testing tool

Example: arrival times of requests at a network-attached storage



Example: HTTP log files in Apache web servers

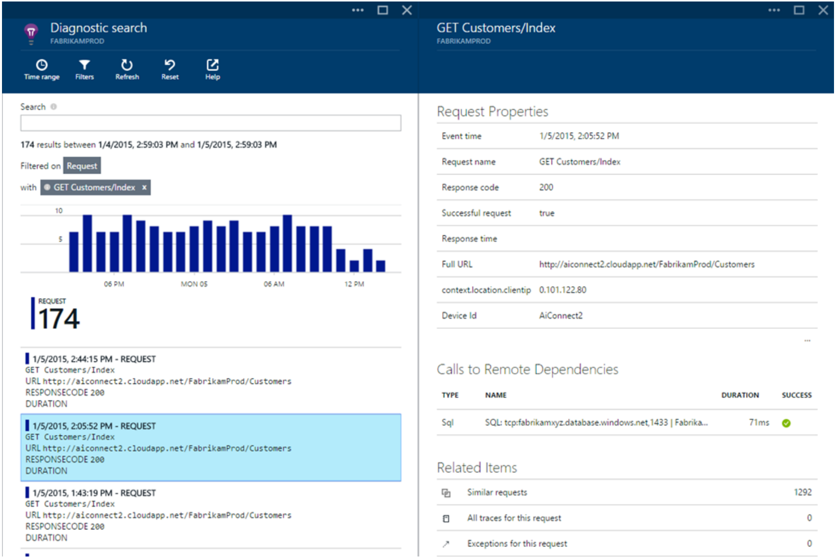
A typical Apache web server access log:

```
125.45.000.166 - - [10/Jun/2010:11:02:34 +0000] "GET http://www.yyy.it/index.html HTTP/1.0" 200 8859
"http://www.google.com/" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
125.45.000.166 - - [10/Jun/2010:11:04:17 +0000] "GET http://www.yyy.it/contact.html HTTP/1.1" 404
1010 "http://www.yyy.it/index.html" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
125.45.000.166 - - [19/Jun/2010:11:04:18 +0000] "GET http://www.yyy.it/contact.gif HTTP/1.1" 200 15890
"-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
...
```

Anatomy of the logfile format:

- `125.45.000.166`: client IP address of the request
- `[10/Jun/2010:11:02:34 +0000]`: timestamp
- `"GET http://www.yyy.it/index.html HTTP/1.0"`: HTTP request (`GET` for downloads, `POST` for forms)
- 200: success code (404 or 500 for errors)
- 8859: size of response in bytes, excluding HTTP headers.
- `"http://www.google.com/"`: referer URL (last page visited)
- `"Mozilla/4.0 ..."`: browser information

Example: Azure AppInsights



Why modelling computer workloads?

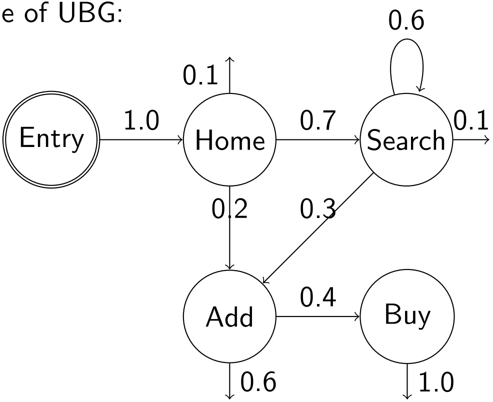
- Workload traces are essential to understand IT system performance, but some common issues limit their use:
  - Privacy: companies don't like to give access (even to their own employees) to logs tracing customer activities.
  - Inflexibility: difficult to manipulate the properties of a trace to study system sensitivity to a workload parameter.
  - Noise: in a trace, we do not always know what is noise and what is not. This can mislead the analysis.
  - Overfitting: what we learn may depend too much on specific trace instances.

Why modelling computer workloads?

- Workload model: a model that can generate traces similar to the ones observed in the system.
  - *e.g.: statistical distributions, Markov chains, automata, ...*
- Workload characterization: model parameters are fitted to traces to capture their essential characteristics.
- Workload models have some advantages compared to traces:
  - Repetitions: models can generate similar, but nevertheless non-identical, workload traces.
  - Understanding: modelling increases our understanding and can lead to system optimizations based on it.
  - Availability: traces are not always available, but they can be generated by models instantiated with artificial parameters.
- We study some models for computer system workloads.

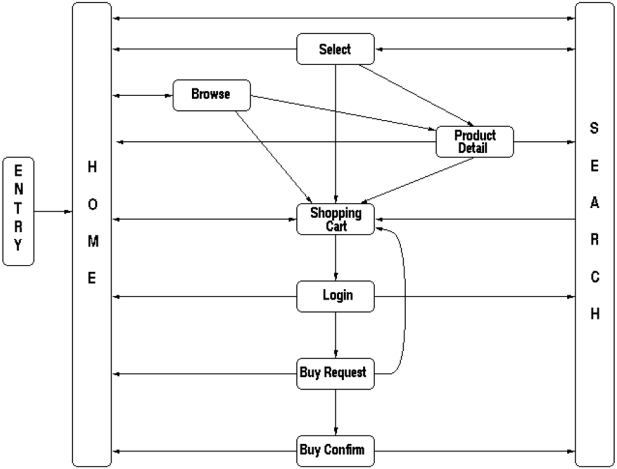
User behaviour graph (UBG)

- User session: the sequence of pages visited by a user.
- UBG: a probabilistic automaton describing user sessions.
  - states = page, session or service invocations
  - arc weights = transition probabilities
  - outgoing arcs = end of the session
- An example of UBG:



Example: UBG in TPC-W benchmark

- TPC-W: A classic benchmark for web servers (now retired).



Source: Menascé

User behaviour graph (UBG)

- UBGs are special discrete-time Markov chains (DTMCs).
  - Users always start from the Entry state.
  - p_{ij} is the transition probability from node i to j
  - After visiting i, the user visits page j with probability p_{ij}.
  - Upon closing the session the user reaches the Exit node (not shown in the UBG).
- We denote by $\mathcal{S}$ the set of DTMC states.

User behaviour graph (UBG)

- Every DTMC is described by a transition probability matrix

$$P = [p_{ij}] = \begin{array}{c c} & \begin{array}{cccccc} E & H & S & A & B & X \end{array} \\ \begin{array}{c} E \\ H \\ S \\ A \\ B \\ X \end{array} & \begin{pmatrix} 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0.2 & 0 & 0.1 \\ 0 & 0 & 0.6 & 0.3 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 0.4 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \end{pmatrix} \end{array}$$

- Note: rows must sum to 1.
- Upon reaching the Exit (X) we never jump to another state. In DTMC theory this is called an absorbing state.

Using UBGs in practice

- UBGs can be fitted on data of a single or multiple client IPs
- p_{ij} values can be initially extracted from log files using the HTTP request and referer URL fields, e.g.,

$$p_{ij} = \frac{\text{number of requests for } j \text{ with } i \text{ as referer URL}}{\sum_{k \in \mathcal{S}} \text{number of requests for } k \text{ with } i \text{ as referer URL}}$$

- Care should be taken when parsing log files:
  - Several users may share the same IP (e.g., users behind a firewall or proxy)
  - A user may navigate with two or more open browser
  - A user may wait several minutes between sending requests, when does a session terminate?
    - A threshold needs to be defined (e.g., 30 minutes).

Using UBGs in practice

- Using UBGs, we can perform:
  - simulation to generate similar, but non-identical, sessions.
    - *e.g., validate the system under varying workload mixes*
  - analysis can help understanding user behaviour.
    - *e.g., understand how website topology affects navigation*
  - modification can help exploring the consequences of changes.
    - *e.g., what if we merge two pages?*
  - clustering can help grouping similar users into classes.
    - *e.g., helpful for business analytics, pre-fetching, sizing, ...*

Simulating a UBG

$i := E$ /* initial state */
while simulation is not over **do**
  print i
  $r :=$ a random number in [0,1)
  for $j \in \mathcal{S}$ **do**
    if $r \leq \sum_{k \leq j} p_{ik}$ **then**
      $i = j$
      break
    end if
  end for
end while

Properties

Visit ratios:

- What is the average number of visits $V(i)$ to state i? (i.e., the mean number of invocations to the corresponding page)
- In the example, $V(\text{Add}) = 0.725$ hence $V(\text{Buy}) = 0.725 \times 0.4 = 0.29$.
- Generalising the argument, we need to solve the linear system:

$$V(E) = 1, \qquad V(j) = \sum_{i \in \mathcal{S}} V(i) p_{ij} \qquad \forall j \in \mathcal{S} \setminus \{E\}$$

where S is the set of states and E=Entry.

Session lengths:

- What is the average session length L for a user?

$$L = \sum_{i \in \mathcal{S} \setminus \{E, X\}} V(i)$$

Properties

Session length distribution:

- How often does the user reach state i after n page visits?
- $\pi^{(n)}(i)$: probability of being in state i at the nth invocation.
- $\pi^{(n)} = [\pi_i^{(n)} | i \in \mathcal{S}]$: state probability vector
- In particular, $\pi^{(0)} = [\pi_E^{(0)}, \pi_H^{(0)}, \dots, \pi_X^{(0)}] = [1, 0, \dots, 0]$
- Similarly to the visits, we need to solve the linear system:

$$\pi_j^{(n)} = \sum_{i \in \mathcal{S}} \pi_i^{(n-1)} p_{ij} \qquad j \in \mathcal{S}$$

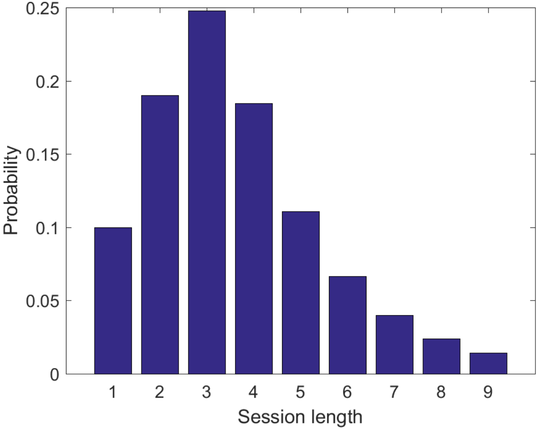
or in matrix form

$$\pi^{(n)} = \pi^{(n-1)} P \qquad \Rightarrow \qquad \boxed{\pi^{(n)} = \pi^{(0)} P^n}$$

Properties

Session length distribution:

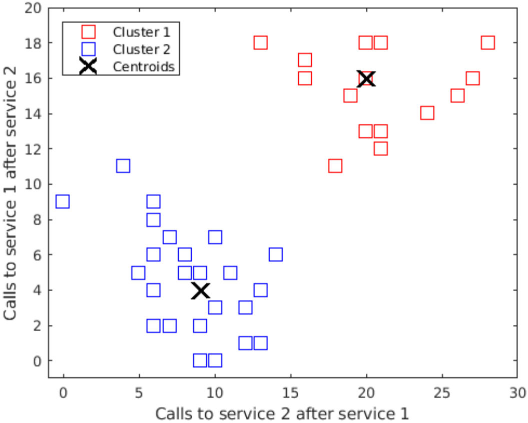
- $\pi_X^{(n)}$ probability of leaving the system in n page visits or less.
- $\pi_X^{(n)} - \pi_X^{(n-1)}$ is thus the probability of completing the session after exactly n page requests.



Fitting UBGs from log data

- UBGs can be fitted automatically from website logs
- Define for a user u the following matrix
  - C_u: entry (i, j) counts the visits to page j right after visiting i
  - After normalizing rows to sum to one, C_u becomes a UBG
- If there are n pages, C_u maps to a point in the Euclidean space with n^2 dimensions
- Each point in this space represents a possible UBG, not necessarily one observed in the logs, but one that can be easily created in load tests or other applications.

Example: centroids as typical user profiles



Each square represents a user. Each centroid models a class of users (cluster) and the associated UBG may be used for load testing, scheduling, load balancing, pre-fetching, personalization, ...

Clustering with the k-means algorithm

- k-means: a classic iterative clustering algorithm
- Input: points in Euclidean space, number of clusters k
- Output: coordinates of k centroids
- Pseudocode:
  1. Initialize centroid positions randomly
  2. Repeat until convergence of the centroid positions:
     - For every point, assign it to the cluster with nearest centroid
     - For every cluster, recalculate the position of the centroid

Clustering with the k-means algorithm

- Distances are quantified using the Euclidean distance

$$d(C_u, C_v) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\big(C_u(i,j) - C_v(i,j)\big)^2}$$

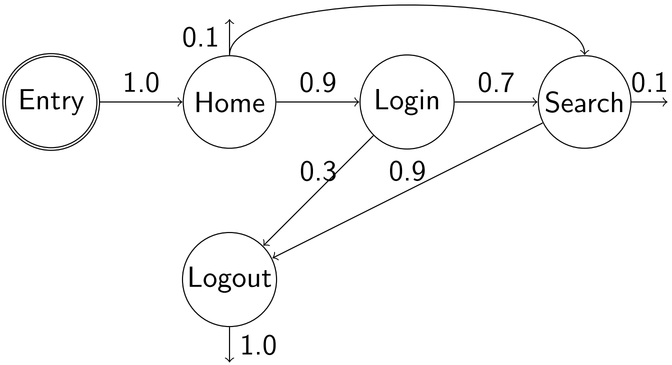
  where $C_u(i,j)$ is the element of C_u in row i and column j.
- For every cluster, centroid positions recalculated by averaging the coordinates of the points within the cluster

$$c(i,j) = \frac{1}{n_c}\sum_{u\in\text{cluster}} C_u(i,j)$$

  where $c(i,j)$ is the centroid coordinate on dimension (i,j) and n_c counts the points currently assigned to the cluster.

Limitations of UBGs

- UBGs do not fully specify interactions, e.g., what to search?
- UBGs are agnostic of resource usage, e.g., CPU, memory, ...
- Some paths might be invalid in the real system, *e.g.* *Home*→*Search*→*Logout* without a *Login*.



Example: UCML - an extension of UBGs

- Different user classes, conditional actions, data flows, ...
- Other variants are integrated as tool-specific languages in load generation tools *(e.g., Jmeter, Load runner, Wessbas, ...)*.