CLOUD 2013

香港中文大學
The Chinese University of Hong Kong

Presented by

Zibin Zheng

zbzheng@cse.cuhk.edu.hk

DR^2: Dynamic Request Routing for Tolerating Latency Variability in Cloud Applications

Jieming Zhu, Zibin Zheng, and Michael R. Lyu

June 28, 2013

Outline

- Introduction
- Main Challenges
- DR^2 Approach
- Experiments
- Conclusion

Introduction

- **Cloud computing**
  - Internet-based virtual computing environment
  - Shared configurable resources: infrastructure, platform, software, etc.
  - Pay-per-use, cost-effective
- **Online cloud applications**
  - Search engine (e.g., Google)
  - Social network (e.g., Facebook)
  - E-commence website (e.g., Amazon)
  - …

Fig. from Google image

Introduction

- **Application latency**
  - Time duration between a request and a response
  - Evaluate the performance of online cloud applications

- **The cost of latency**
  - 0.5s delay: 20% drop in Google's traffic
  - 0.1s delay: 1% drop of Amazon's sales.

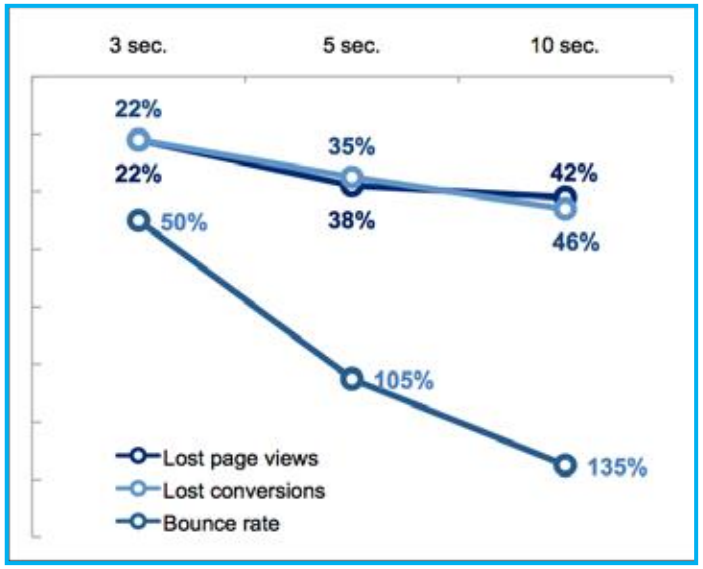
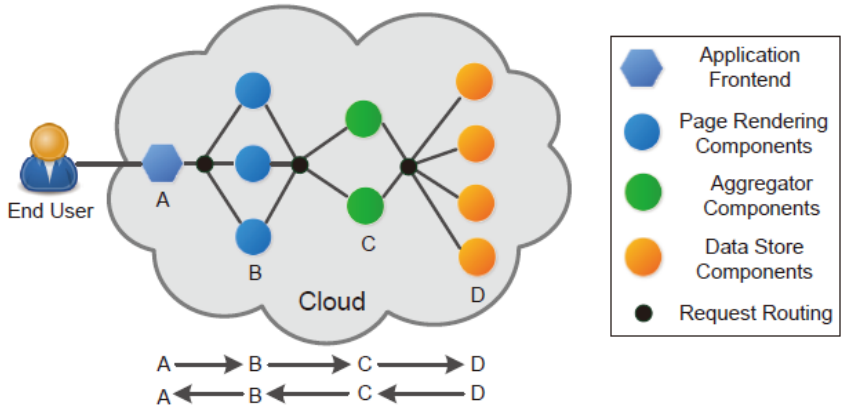


Fig. from Interxion's whitepaper

Introduction

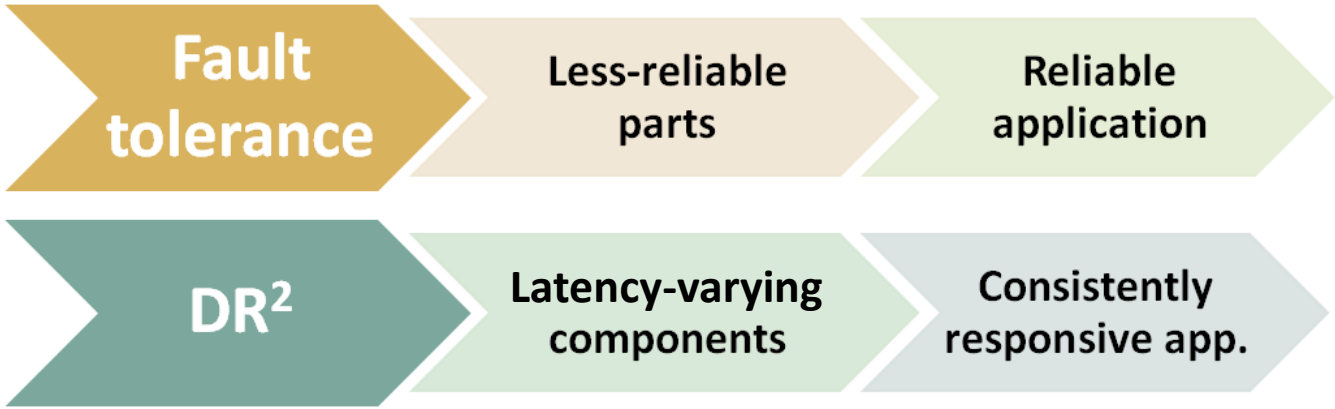
- **However, users perceive variability on latency, due to:**
  - Applications: involve numerous cloud components
  - Scaling up: components deployed **across data centers**
  - Relying on the Internet for connectivity
- **Application request example**



Example from Amazon's page request

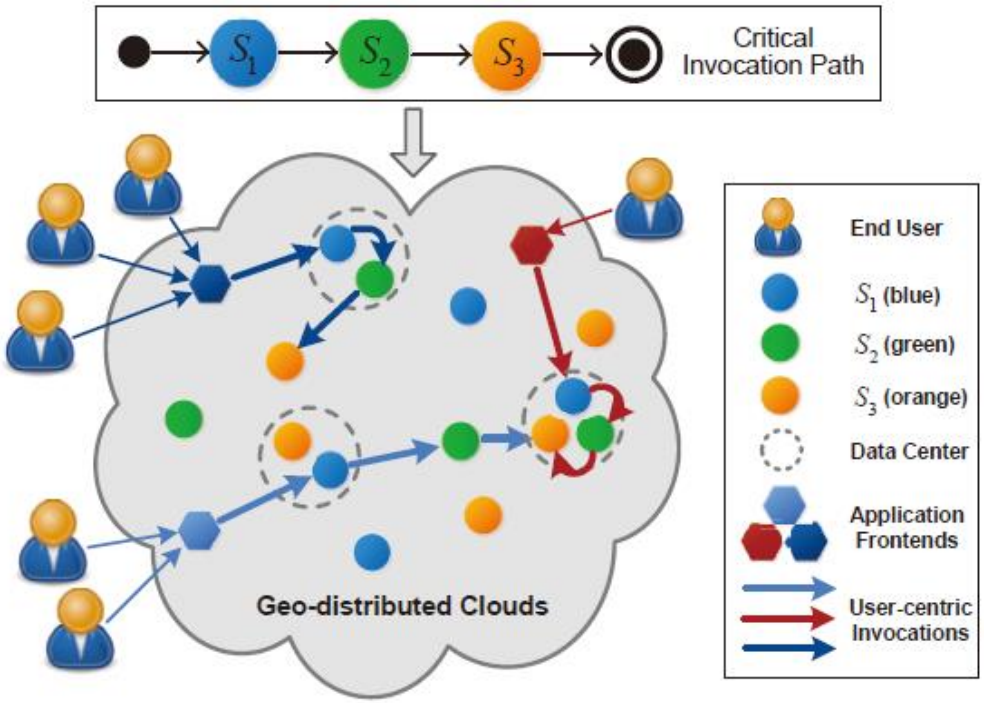
Introduction

- **How to build consistently low-latency cloud applications, with**
  - Geo-distributed cloud components
  - Varying latency between components
- **Our proposal : Dynamic Request Routing (DR^2)**
  - Take advantage of redundant components
    - E.g., much redundancy for fault tolerance / load balancing



Introduction

- **A prototype of DR²**

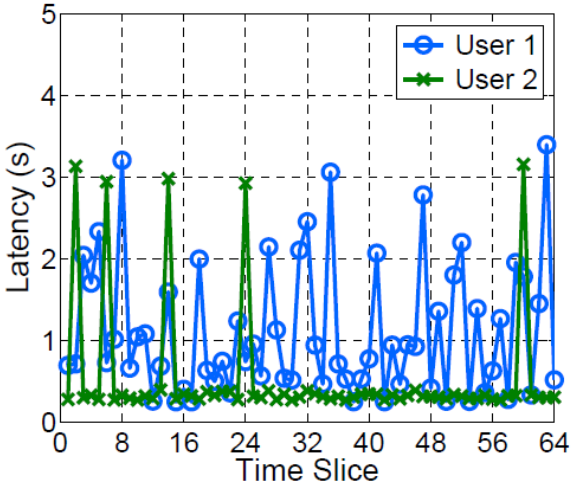


Dynamically rout the requests to different components with timely latency minimization

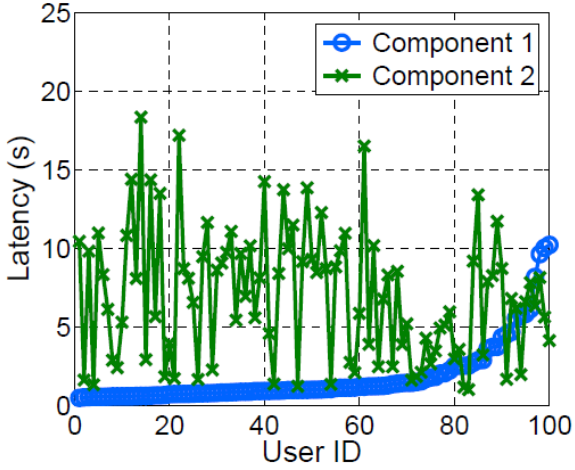
Challenges

Challenges

- **Latency variability**
  - Relying on the Internet for connectivity
  - Fluctuations over time
- **Adaptivity**
  - Adaptive to the latency dynamics
- **User centricity**
  - Optimize the request for each single user
- **Scalability**
  - Scalable and efficient



(a) Latency v.s. Time Slice

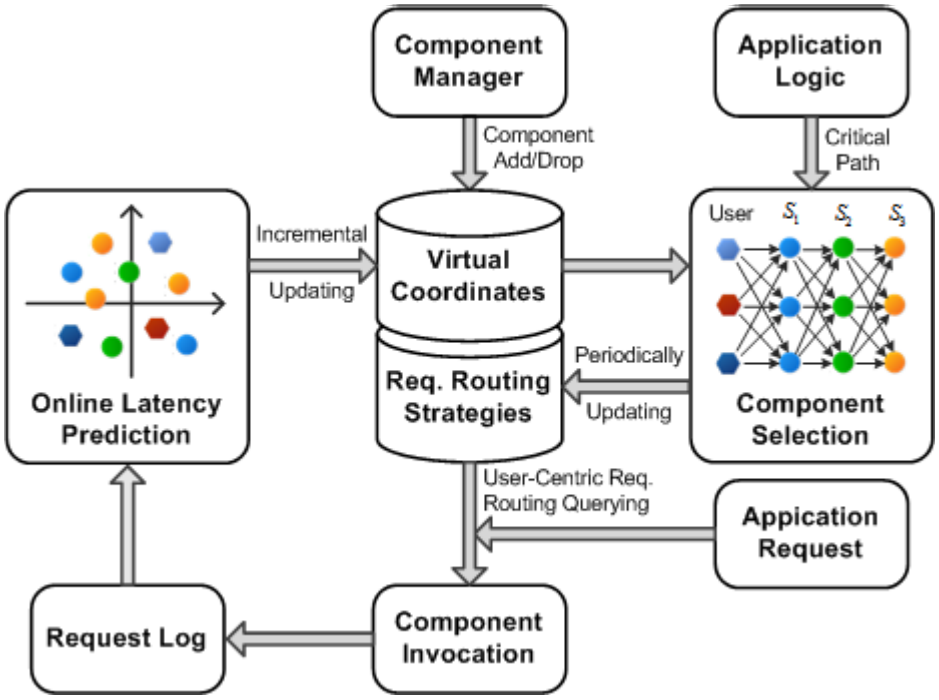


(b) Latency v.s. User ID

DR^2 Approach

System Architecture

- **DR^2: Dynamic Request Routing Framework**
  - Phase 1: Online latency prediction
  - Phase 2: Adaptive component selection



The framework of DR^2

DR^2: Dynamic Request Routing

- **Phase 1: Online latency prediction**
  - Matrix factorization model:

	S_1	S_2	S_3	S_4
U_1	0.2	?	0.3	?
U_2	?	0.3	?	0.4
U_3	0.4	0.6	?	?
U_4	?	?	0.8	0.6

(a) L^U Matrix

	S_1	S_2	S_3	S_4
S_1	0	?	0.7	?
S_2	?	0	?	0.8
S_3	0.5	0.6	0	?
S_4	?	0.4	0.3	0

(b) L^S Matrix

$$\min \Psi(U,S,V) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}^{U}(L_{ij}^{U} - U_i' S_j)^2 + \frac{1}{2}\sum_{k=1}^{m}\sum_{h=1}^{m} I_{kh}^{S}(L_{kh}^{S} - V_k' S_h)^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_S}{2}\|S\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2$$

Squared sum of errors

Regularization term

DR^2: Dynamic Request Routing

- **Phase 1: Online latency prediction**
  - Matrix factorization model
  - Incremental updating of virtual coordinates (continously)

$$(u_i,\ s_j,\ L_{ij}^U) \longrightarrow \psi(U_i, S_j) = \frac{1}{2}(L_{ij}^U - U_i' S_j)^2 + \frac{\lambda_u}{2}\|U_i\|_2^2 + \frac{\lambda_s}{2}\|S_j\|_2^2,$$

$$(s_k,\ s_h,\ L_{kh}^S) \longrightarrow \psi(V_k, S_h) = \frac{1}{2}(L_{kh}^S - V_k' S_h)^2 + \frac{\lambda_s}{2}\|S_h\|_2^2 + \frac{\lambda_v}{2}\|V_k\|_2^2.$$

Algorithm 1: Online Latency Prediction Algorithm

Input: Latency data: L_{ij}^U, L_{kh}^S
Output: The virtual coordinates: U_i, S_j, V_k

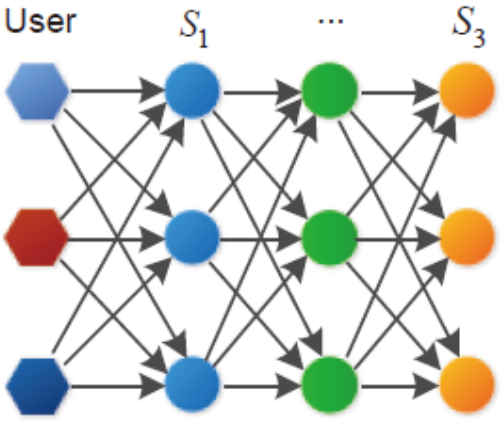
Randomly initialize $U \in \mathbb{R}^{d\times n}$, and $S, V \in \mathbb{R}^{d\times m}$;
repeat /* Incremental updating */
  Collect historical latency data;
  if *receive a latency data sample* (u_i, s_j, L_{ij}^U) **then**
    $U_i \leftarrow U_i - \eta((U_i^T S_j - L_{ij}^U)S_j + \lambda_u U_i)$;
    $S_j \leftarrow S_j - \eta((U_i^T S_j - L_{ij}^U)U_i + \lambda_s S_j)$;
  else if *receive a latency data sample* (s_k, s_h, L_{kh}^S) **then**
    $S_h \leftarrow S_h - \eta((V_k^T S_h - L_{kh}^S)S_h + \lambda_v V_k)$;
    $V_k \leftarrow V_k - \eta((V_k^T S_h - L_{kh}^S)V_k + \lambda_s S_h)$;
until *converge*;

Update the virtual coordinates U_i, S_j

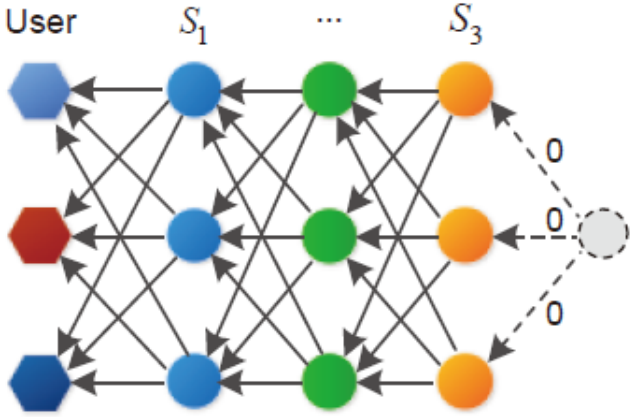
Update the virtual coordinates S_h, V_k

DR^2: Dynamic Request Routing

- **Phase 2: Adaptive component selection**
  - **Problem formulation**
    - Nodes: users and components
    - Edges: available invocations
    - Weights: predicted latencies
    - Find shortest path for each user
  - **Straightforward point-point Dijkstra computation**
    - Not efficient
  - **Proposed solution**
    - Get all the shortest paths in one traverse



(a) Invocation Graph



(b) Virtual Graph

DR^2: Dynamic Request Routing

- **Phase 2: Adaptive component selection**
  - Problem formulation
  - Shortest path computation (periodically update)

Algorithm 2: Adaptive Component Selection

Input: Critical Path, Virtual coordinates: U_i, S_j, V_k
Output: Component selection strategy

```
1  Construct the virtual graph VG based on the critical path;
2  Topologically sort VG to VG_list;
3  foreach node v in VG_list do /* Initialization */
4      if v ∈ user then
5          v.out ← U_i; v.in ← none;
6      else v.out ← V_k; v.in ← S_j;
7      if v is in the last level of the critical path then
8          v.latency ← 0;
9      else v.latency ← inf;
10     v.parent ← none;
11 foreach node v in VG_list do
12     foreach node w in adjacency of v do
13         if w.latency lg v.latency + w.out'*v.in then
14             w.latency ← v.latency + w.out'*v.in;
15             w.parent ← v;
16 foreach node v ∈ user do /* Output the selection
   strategy v.path */
17     add v.parent to v.path;
```

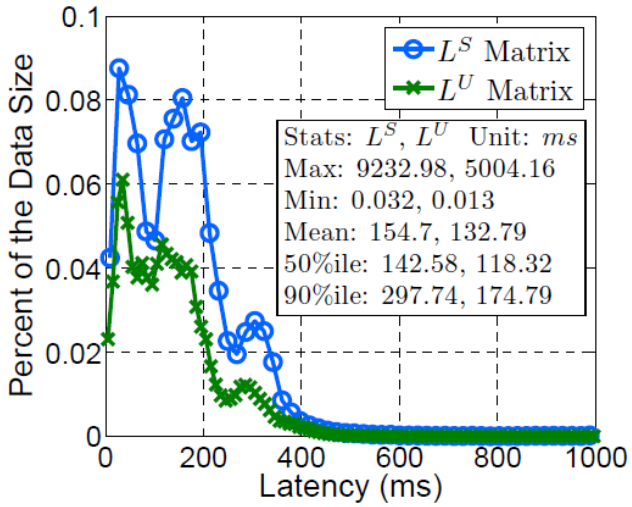
Convert the original graph to the virtual graph (VG)

Fast shortest path algorithm on DAG with linear complexity of O(n+m)

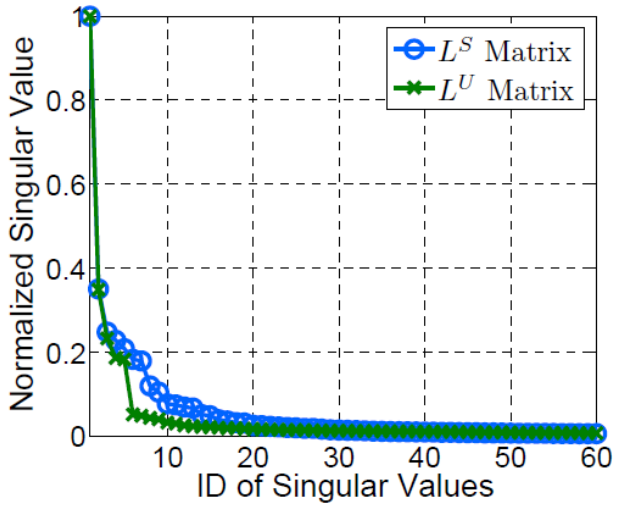
Experiments

Experiments

- **Dataset description**
  - **Dataset1:** measured by ourselves
    - 1350 × 460 L^U, 460 × 460 L^S
  - **Dataset2:** extracted from [Zhang et. al 2011]
    - 4532 users × 142 components × 64 time slices
  - **Dataset3:** synthetic dataset



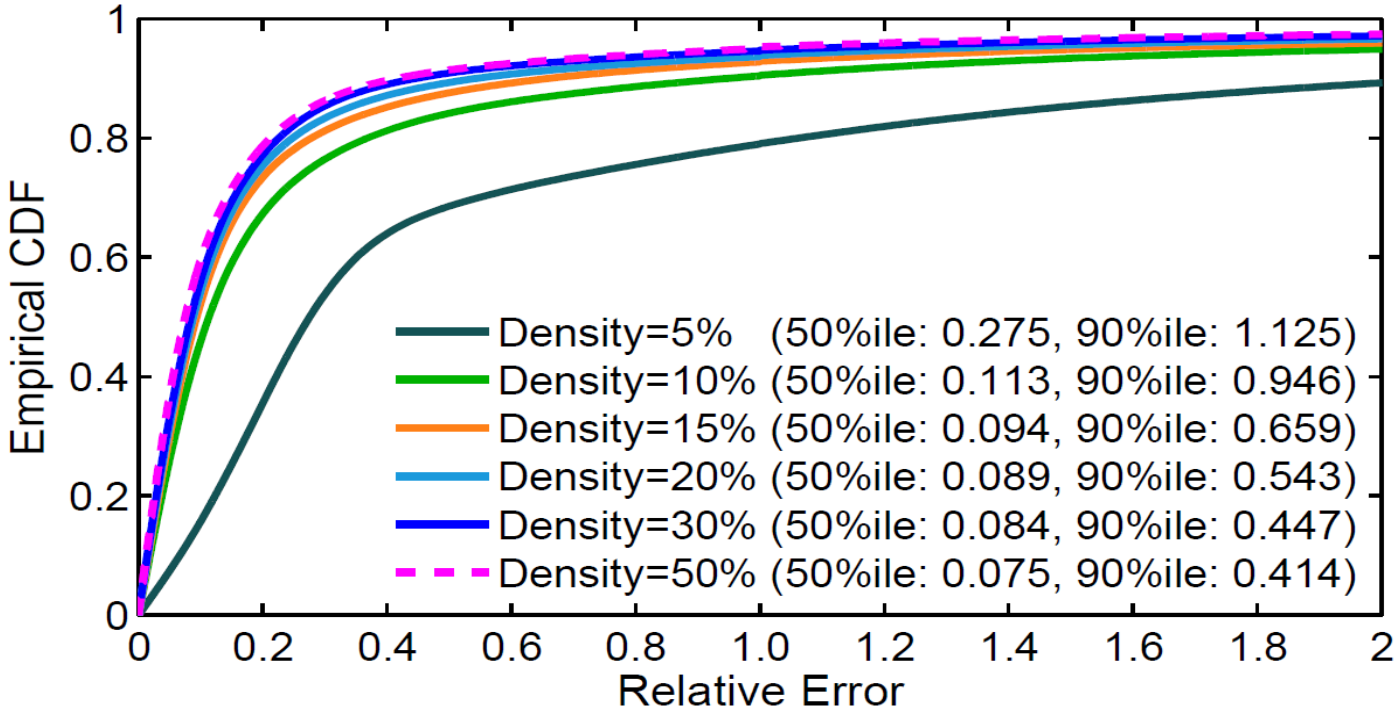
(a) Data Statistics



(b) Singular Values

Performance Evaluation

- **Accuracy of online latency prediction**
  - Accuracy improves with the increasement of matrix density
  - But the increasement diminishes



Performance Evaluation

- **Performance comparison**
  - **Random**: randomly select the candidate component
  - **Greedy-M**: select the best component at each step
  - **DR²**: our approach

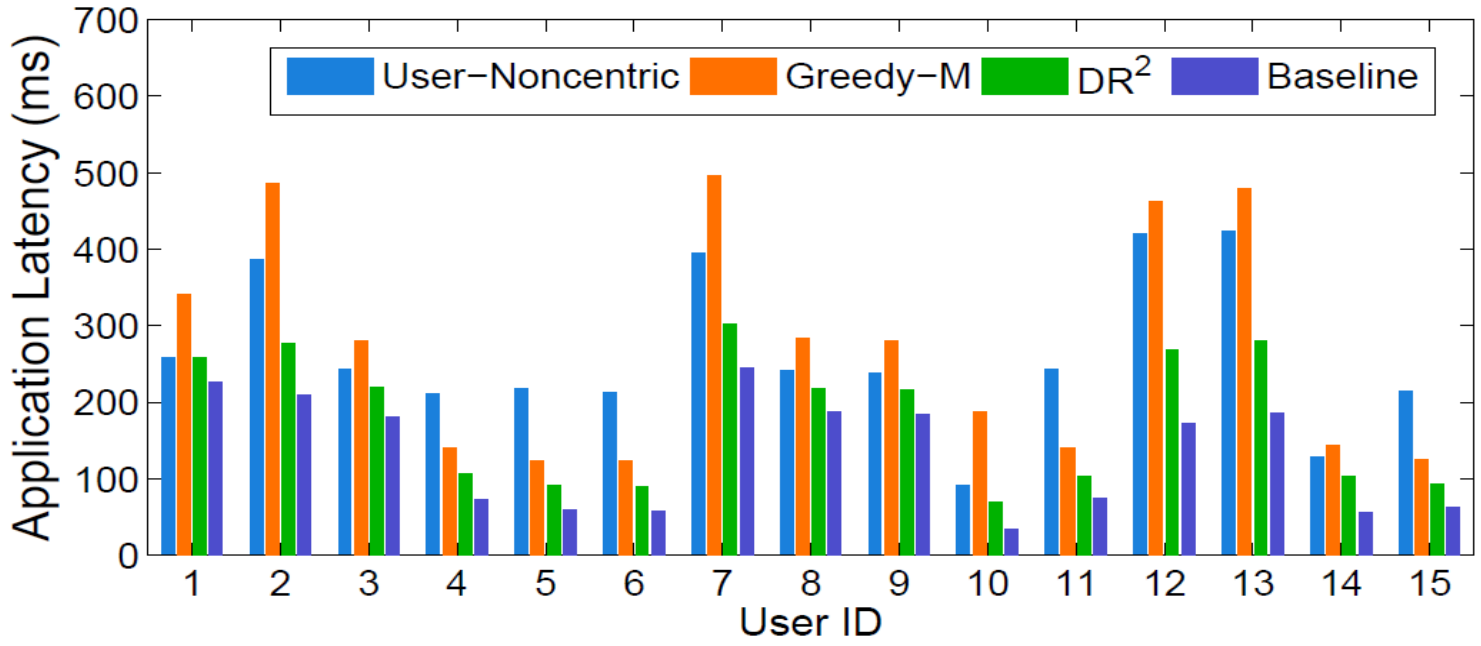
TABLE I. PERFORMANCE COMPARISON

Methods	Density = 10%	Density = 20%	Density = 30%	Density = 40%	Density = 50%	Density = 100%
	$ARE \pm std$	$ARE \pm std$	$ARE \pm std$	$ARE \pm std$	$ARE \pm std$	ARE
Random	6.444 ± 0.088	6.446 ± 0.047	6.435 ± 0.043	6.431 ± 0.035	6.405 ± 0.069	6.436
Greedy-M	0.888 ± 0.194	0.613 ± 0.086	0.517 ± 0.110	0.506 ± 0.087	0.496 ± 0.092	0.656
DR²	**0.412 ± 0.108**	**0.269 ± 0.045**	**0.163 ± 0.043**	**0.129 ± 0.020**	**0.089 ± 0.025**	**0**

DR² performs the best, and is close to the baseline

Performance Evaluation

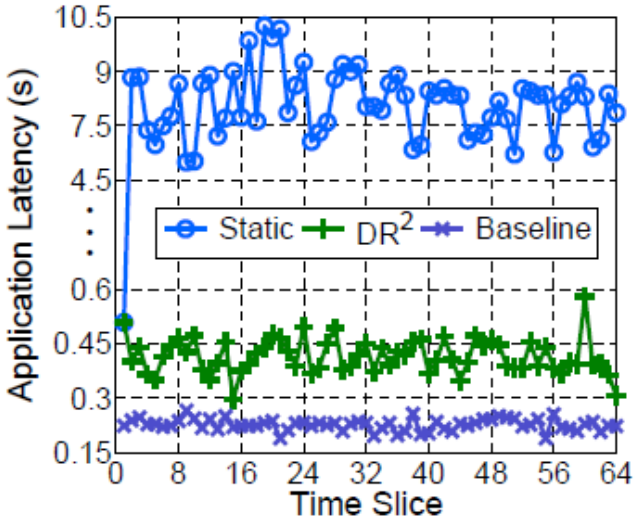
- **Performance on multiple users**
  - Randomly select 15 users as examples
  - User-Noncentric/Greedy-M/DR^2/Baseline



DR^2 optimize the component selection for each user

Performance Evaluation

- **Performance on multiple time slices**
  - **Static:** Not updating the component selection over time
  - **DR²:** Our approach (adaptive component selection)
  - **Baseline:** Using the exact latency data

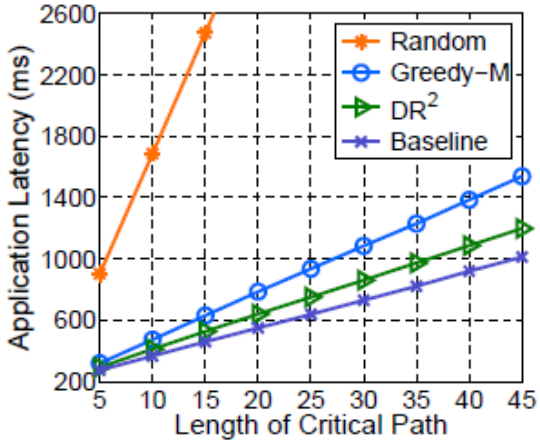


(a) Peformance on Multi. Time Slices

DR² adapts to the dynamics and tolerates the latency variability

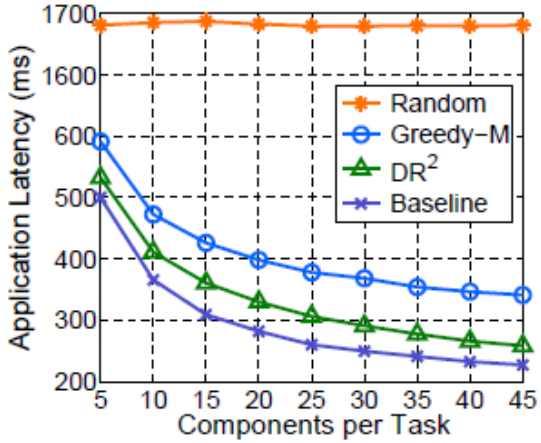
Performance Evaluation

- **Impact of parameters**
  - Length of critical path
  - Components per task
  - Matrix density



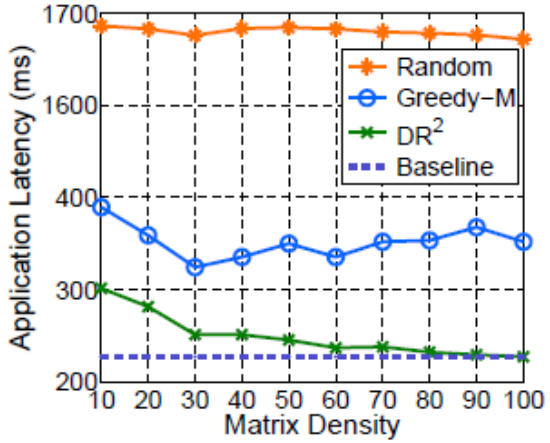
(b) Impact: Length of Critical Path

User num. = 1350,
Componet num. = 10,
Matrix density = 30%



(c) Impact: Components per Task

User num. = 1350,
Criti. Path Length = 10,
Matrix density = 30%

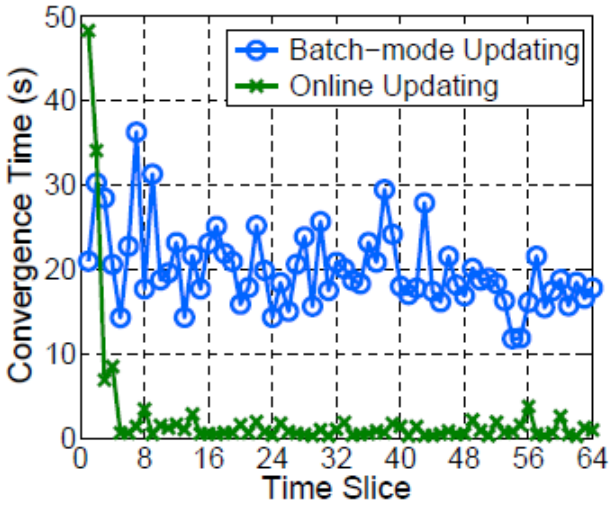


(d) Impact: Matrix Density

User num. = 1350,
Criti. Path Length = 10,
Componet num. = 45

Efficiency Analysis

- **Convergence time of DR²**
  - Batch-mode updating: periodically
  - Online updating: continuously



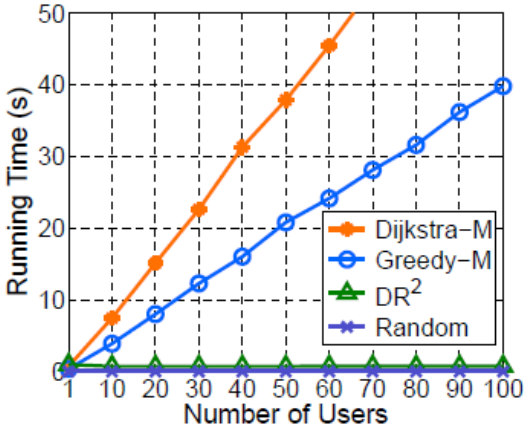
(a) Convergence Time of DR²

DR² converges faster

Efficiency Analysis

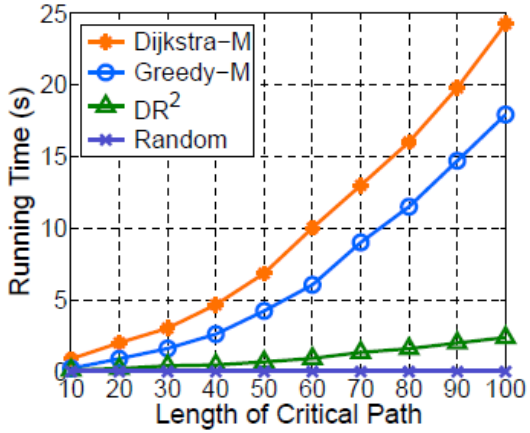
- Scalability
  - Number of users
  - Length of the critical path
  - Component number

Good scalability



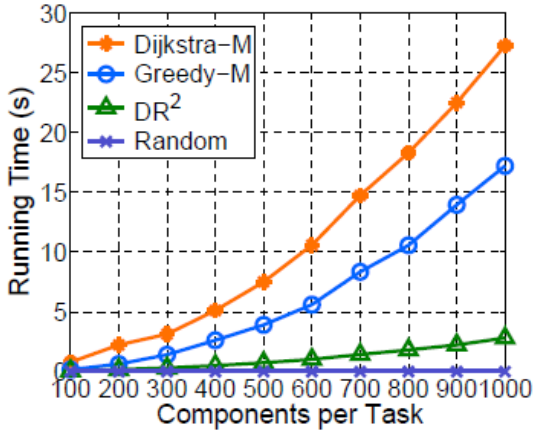
(b) Running Time v.s. Num. of Users

Criti. Path Length = 10,
Componet num. = 500



(c) Running Time v.s. Crit. Path Leng.

Users num. = 10,
Component num. = 100



(d) Running Time v.s. Compon. Num.

User num. = 10,
Criti. Path Length = 10

Conclusion & Future Work

Conclusion

- **DR^2: Dynamic request routing framework**
  - Tolerating latency variability in online cloud applications
  - Extensive experimental results for evaluation
  - Effective, adaptive, user-centric and scalable
- **Future Work**
  - Extend the current framework to consider load balancing strategy
  - Collect more realistic application data for our experimentation

Thank you!

Dataset available:
http://www.wsdream.net