Measuring and Managing Answer Quality in Online Data-Intensive Systems

Jaimie Kelley*, Christopher Stewart*, Devesh Tiwari+, Yuxiong He**, Sameh Elnikety** and **Nathaniel Morris***

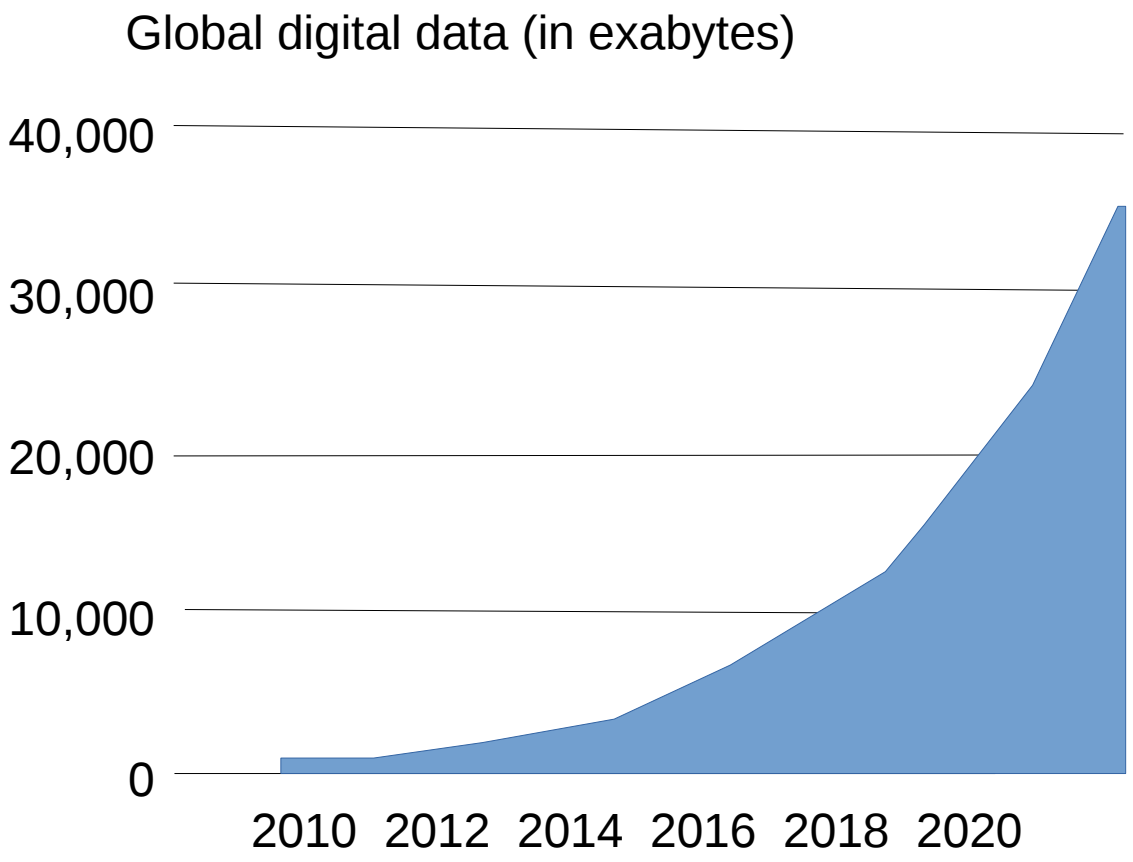
*The Ohio State University, Computer Science and Engineering

+Oak Ridge National Laboratory

**Microsoft Research

Global digital data is expected to grow 30,000X in 10 years.

Source: IDC



“[there is] poor financial justification for merely incrementing online storage. Viable alternates to hanging new disk include:

Implementing tiered storage systems that cost-effectively balance levels of data utility with data availability

....

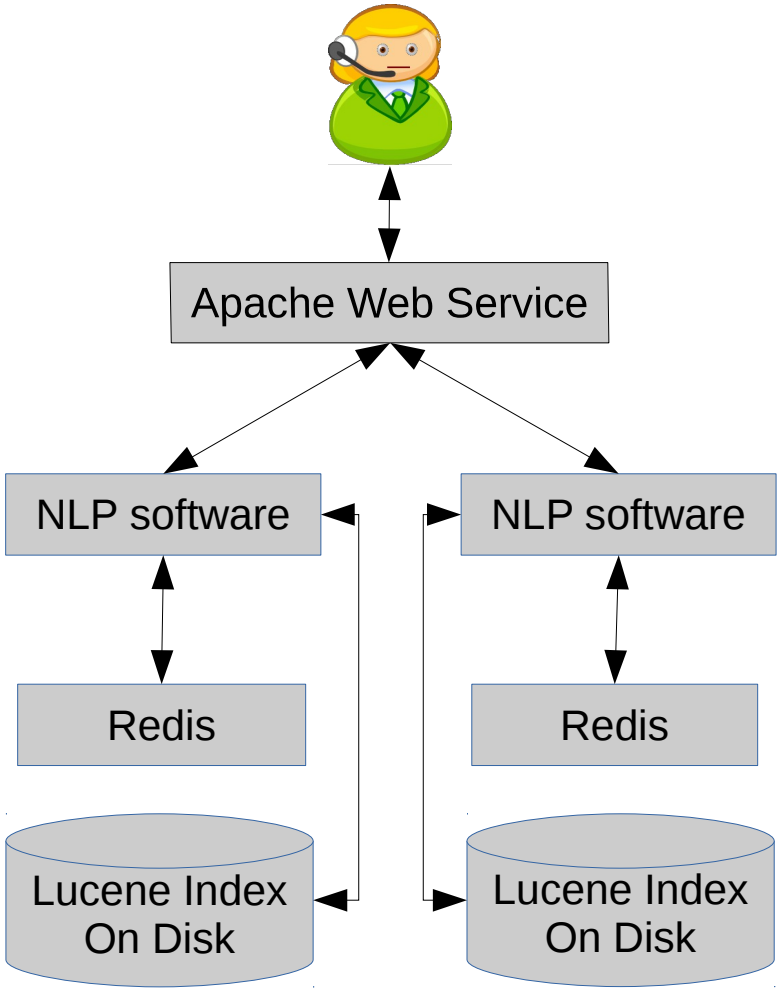
limiting certain analytic structures to **a percentage of statistically valid sample data**”

Doug Laney, Research Vice President
Gartner Research

Online, Data-Intensive Services

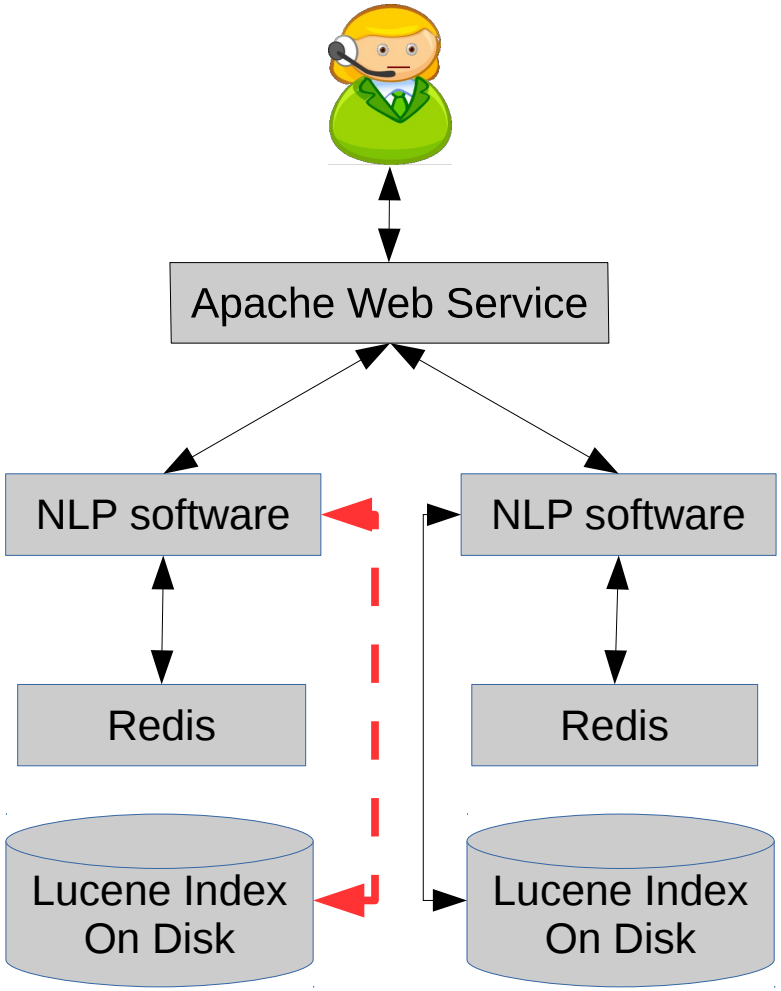
- **Parallelized execution**
- **Distributed software components**
- **Interactive response times**

- Examples:
  - Search engines: Google, Bing, Apache Lucene
  - Recommendation engines: Netflix
  - Question answering systems: OpenEphyra



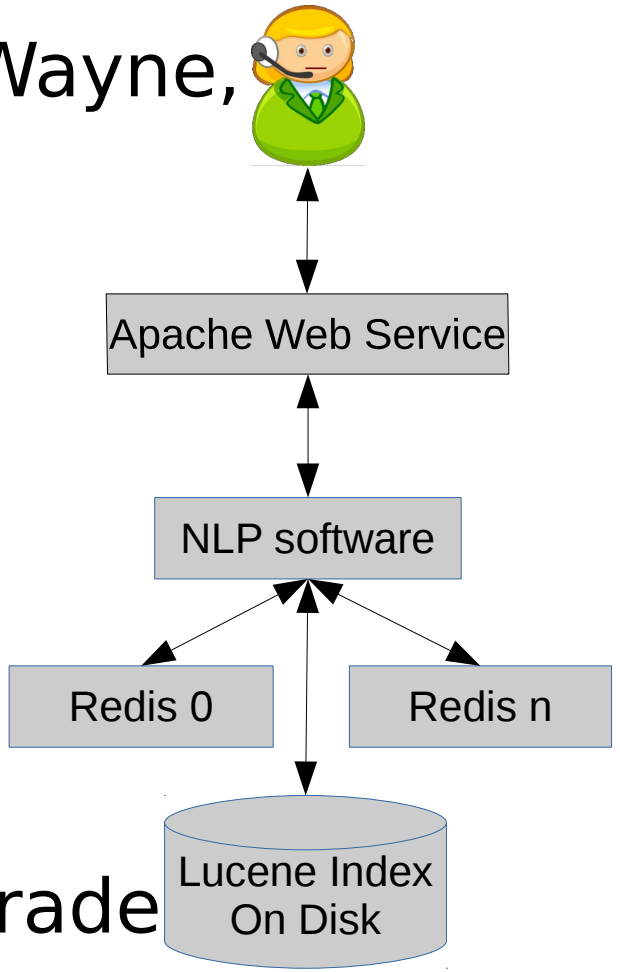
Online, Data-Intensive Services

- To achieve, interactive response times
  - Answers return before slow components finish.
  - **Answers are best effort**
- Problem:
  - Data from slow components could lead to better answers.



OpenEphyra Service

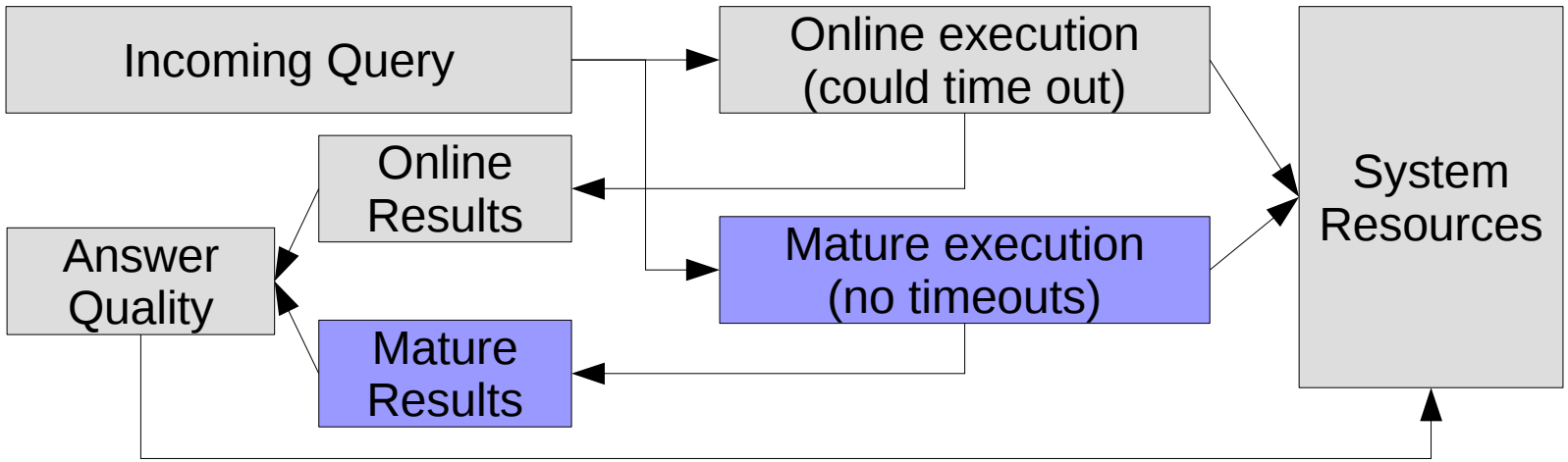
- 15 second answer: Thomas Wayne,
- All data: Bruce Wayne



- Jeopardy Outreach:
  - OpenEphyra vs. 6th-8th grade

Answer Quality

- Application-Defined Similarity function
- Online answer set
- **Mature** answer set (includes all relevant data)



Mature Overlap

- Acquiring answer quality online is time and resource intensive
  - Mature execution uses more resources than online
  - Introduces overhead on all concurrent queries
- Contribution: Overlap mature execution with online execution
  - Use the outputs from online computations to reduce resources needed for mature execution

Design Goals

- **Transparency**: Require no code changes to software components.
  - Use a middleware framework.
- **Timeliness**: Achieve mature results quickly to manage resources.
  - Enable answer quality measurement from an online environment.
- **Low Cost**: Require no increase in allotted resources.
  - Only use the resources currently available to the service.
- **Low Overhead**: Reduce slowdown in online queries.
  - Do not impose undue consequences on the host service.

Transparent Design

- Goal: **Transparency**
  - Challenge: Manage query context (online, mature) without changing the application or communication patterns.
- Framework:
  - Use TCP connection sequence numbers to identify related interactions.
  - Copy network communication to cache.
  - Use request data with ID as key and response as value in key-value store

Design with OS Context Management

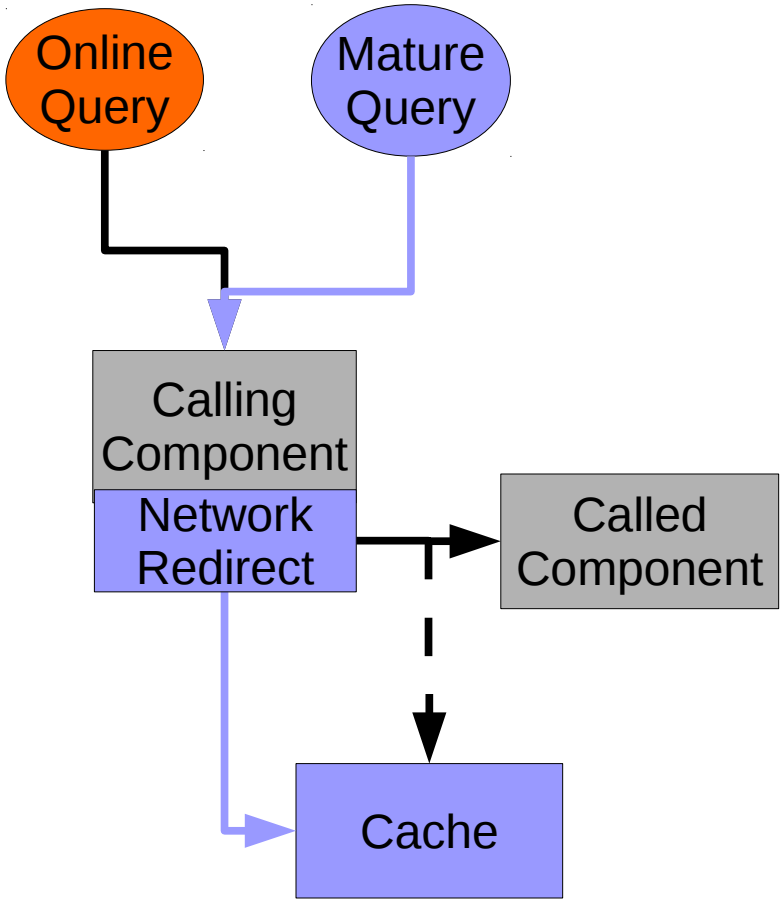
- Messages in thread local memory are annotated with context and query ID
- We set mode according to context and query ID
  - Record (sample online query)
  - Replay (mature execution)
  - Normal (non-sampled online query)
- We use request data with ID as key in cache

Design without OS Context Management

- Global context variable (Record, Replay, Normal) is applied to all concurrent queries
  - All modes provide valid output
  - All concurrent queries use same context
- Propagate context using just-in-time UDP messages
  - Send only to accessed components
  - Timeout modes locally
  - Always reverts to normal
- Message and query ID are used to key cached data

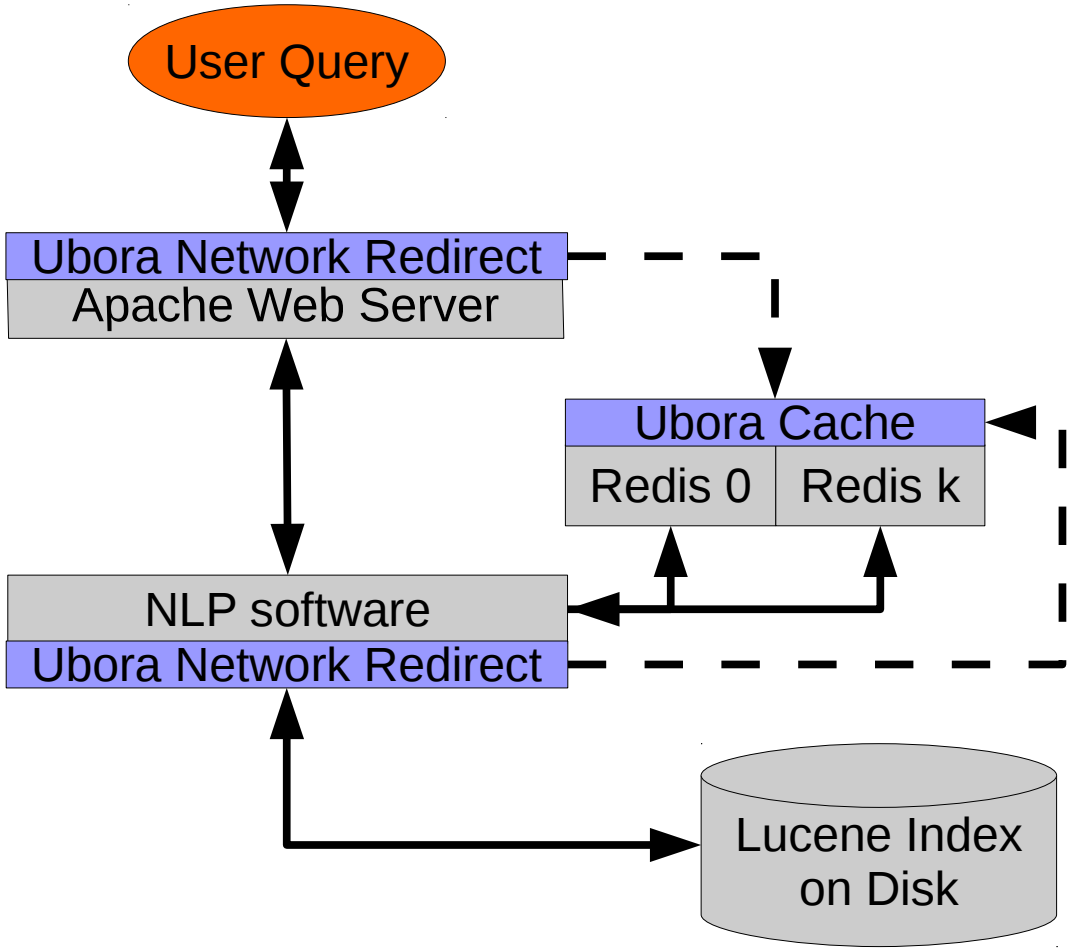
Ubora Implementation Overview

- Records network traffic between components
  - Allows timeout at Calling Component
  - Records after timeout until Called Component completion
- Reissues captured query
  - Requests to Called Component are served by Ubora Cache
  - All data is processed



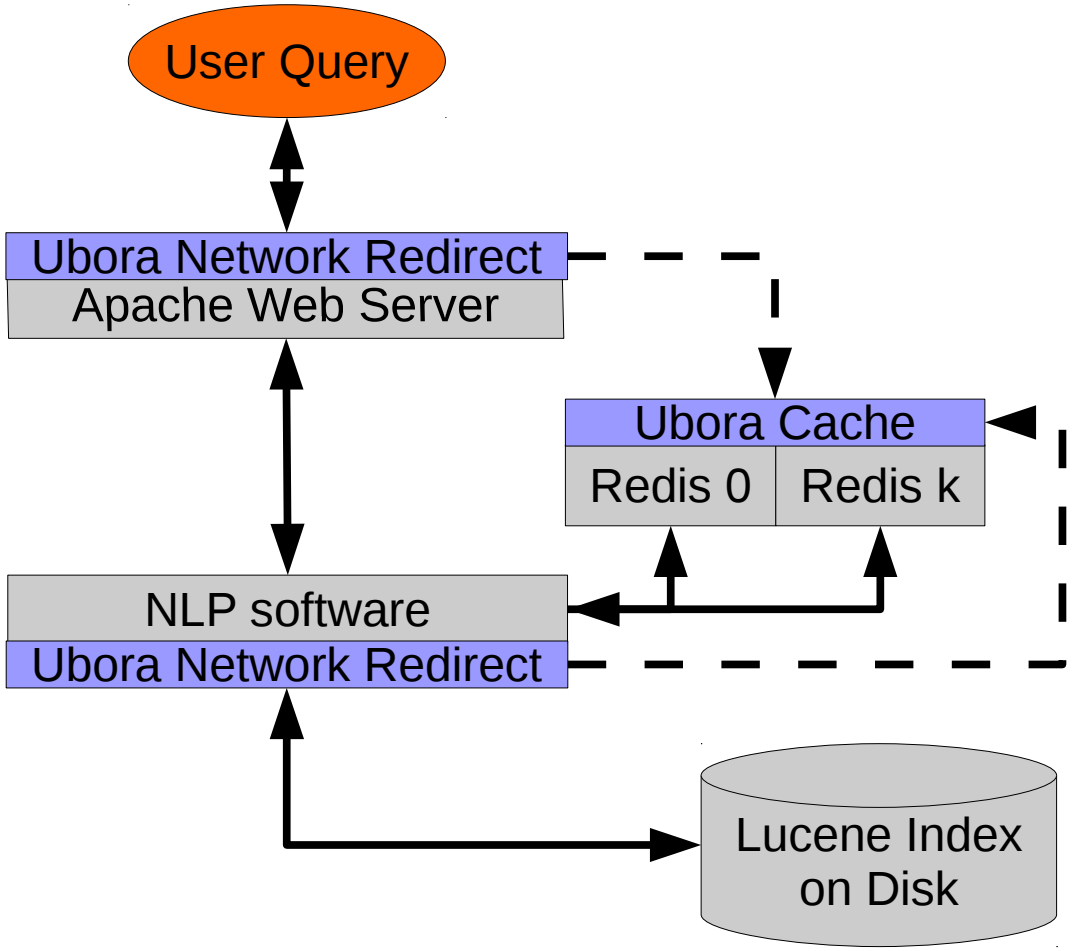
Ubora: A Closer Look at Record

1. User Query arrives at Apache
1. Ubora sends User Query to cache
2. Apache sends query to NLP



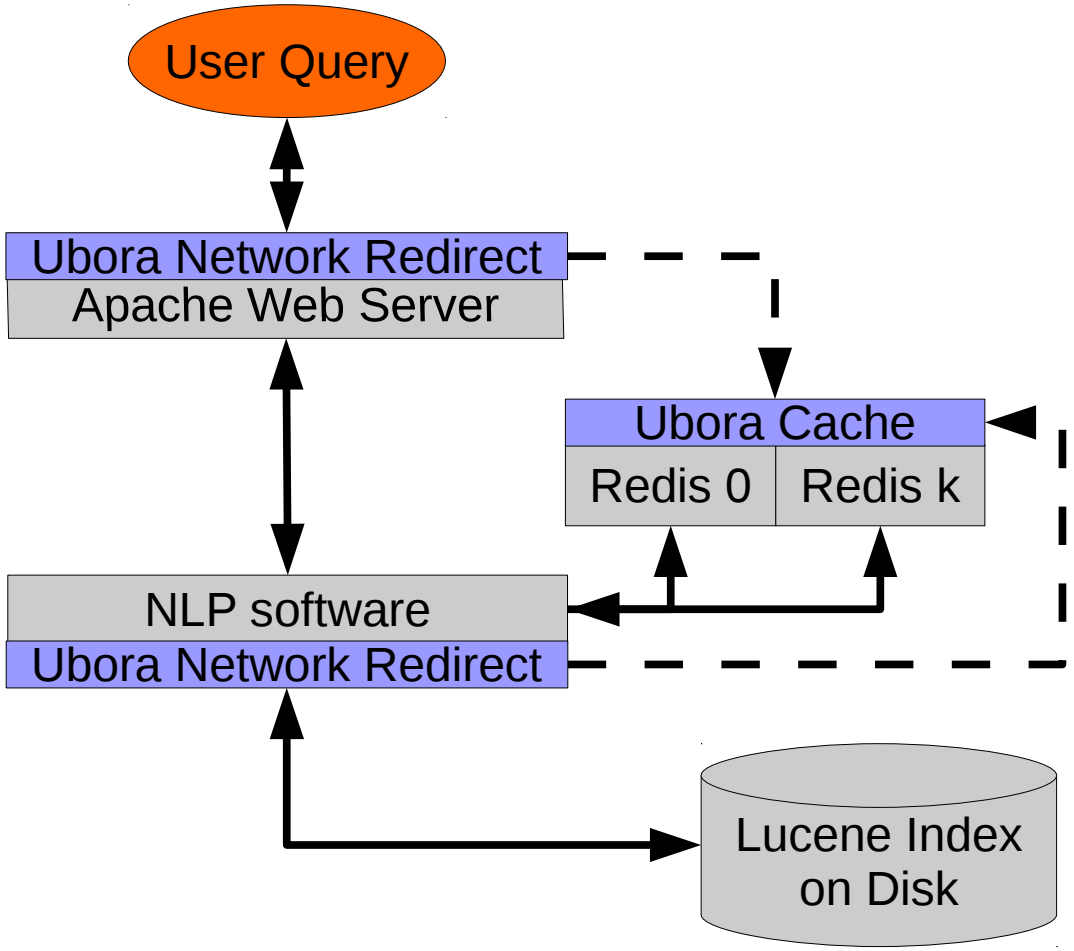
Ubora: A Closer Look at Record

3. NLP queries
   Redis 0...k
3. NLP queries Disk
3. Ubora records Disk queries
4. Redis 0...k answers
   NLP software



Ubora: A Closer Look at Record

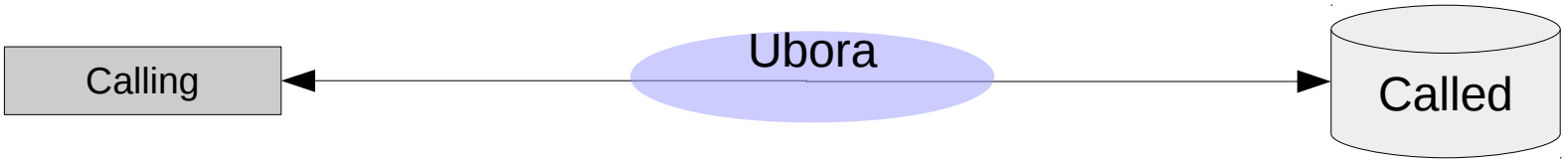
5. Disk answers
   NLP software
5. Ubora records Disk answers
6. NLP times out Disk
6. Ubora interferes.



Record: IPQueue

- iptables
  - Most common use case is network administration
  - Identifying interactions can be time-consuming
  - iptables -A INPUT -p tcp --dport 1064 -j QUEUE
- IPQueue
  - C language extension to allow packet processing
  - Enforces in-order packet processing
  - Allows programmer to arbitrarily change a packet

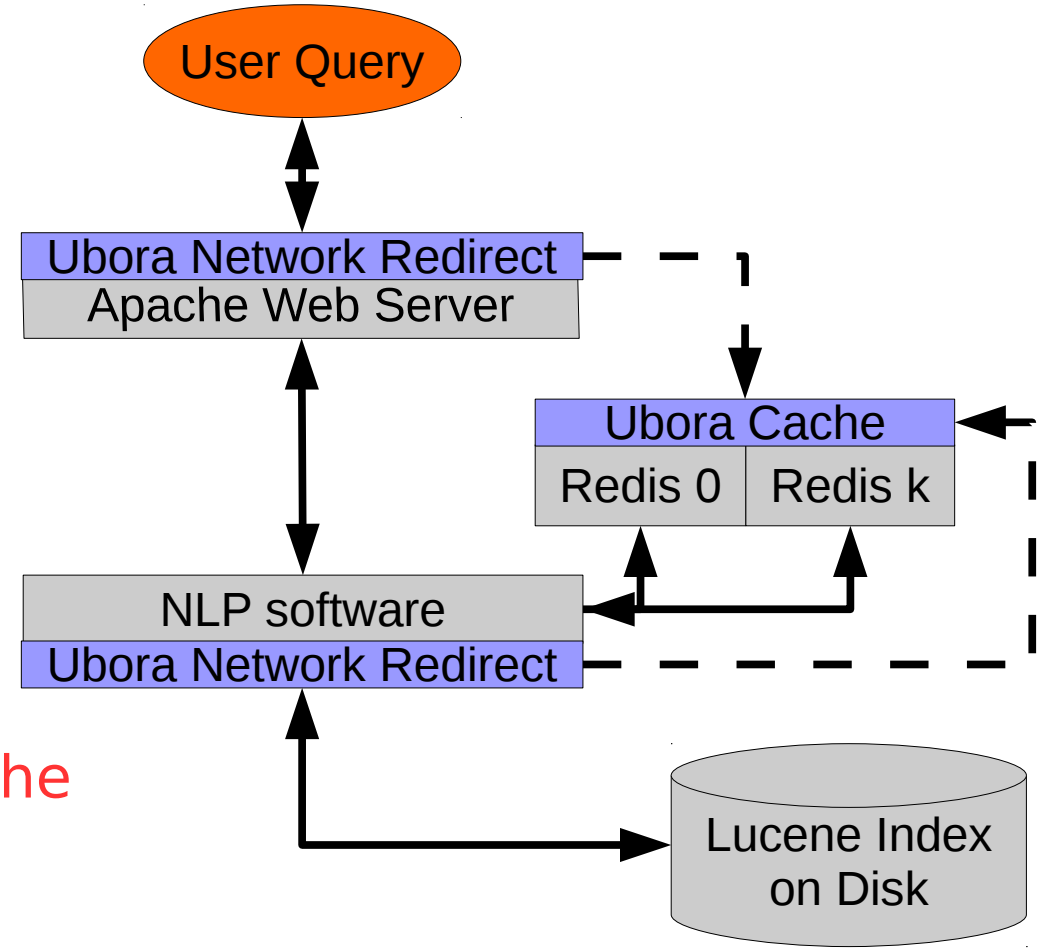
Replace FIN with ACK



- Mangle FIN Packet from Worker:
  - Change tcp->ack = 1
  - Set tcp->fin = 0
  - Replace tcp->acknum
  - Remove payload
  - Compute tcp header checksum
  - Compute ip header checksum
- Drop Additional Data Packets from Archival Storage:
  - status = ipq_set_verdict(h, m->packet_id, NF_DROP, 0, NULL);

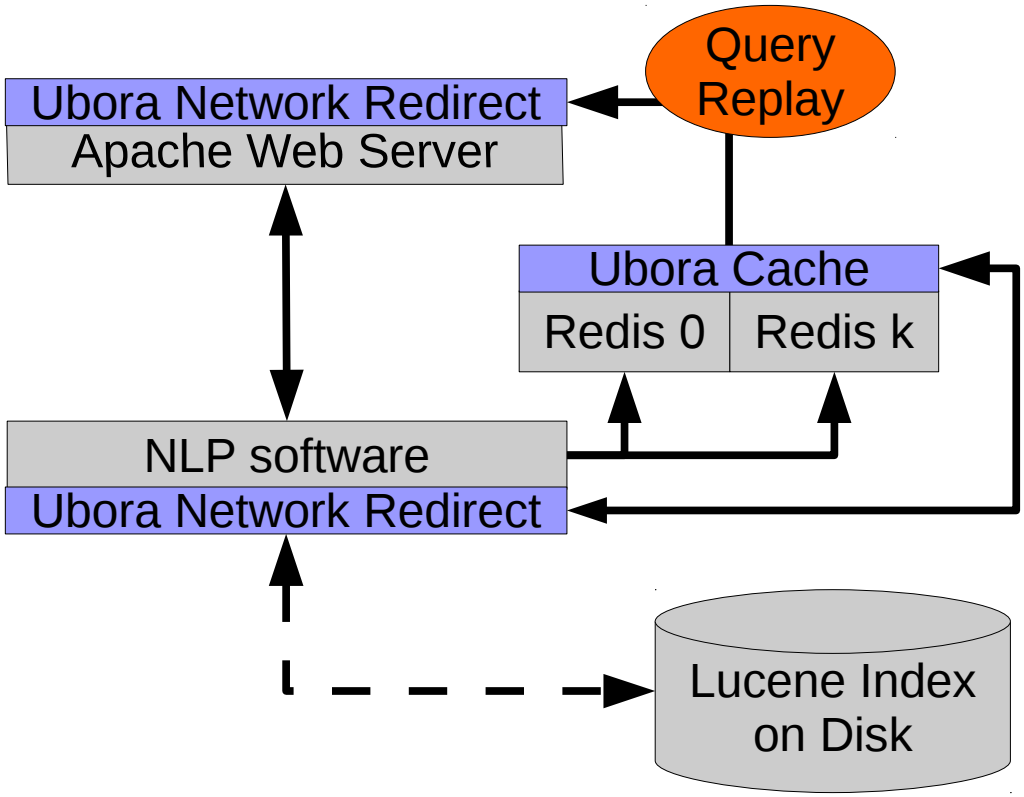
Ubora: A Closer Look at Record

6. Ubora interferes.
7. NLP answers
   Apache
7. Ubora sends key value pair to Ubora Cache
8. Apache answers User
8. Ubora sends Online
   Results to Ubora Cache



Ubora: A Mature Execution

1. Ubora issues query to Apache
2. Apache sends query to NLP
3. NLP queries Redis
3. NLP queries Disk
3. Ubora interferes with Disk access.

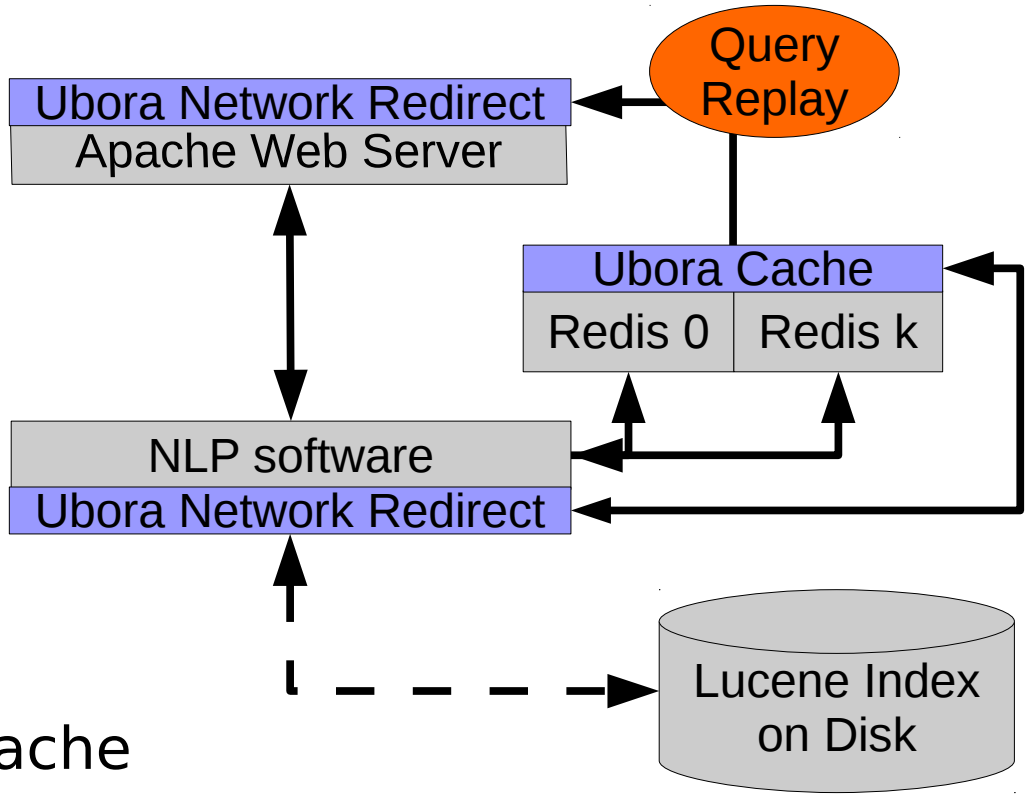


Replay: Man-In-The-Middle

- iptables -t nat -I OUTPUT 1 -p tcp --dport 1064 ! --sport 1066 -j REDIRECT --to-port 1061
  - Port on which Called Component listens: 1064
  - Port on which Man-In-The-Middle listens: 1061
- Man-In-The-Middle binds to specific local ports (like 1066) when connecting to Called Component on 1064.
  - Man-In-The-Middle connects to Called Component when necessary for program correctness
  - Else, blocks all connections to Called Component

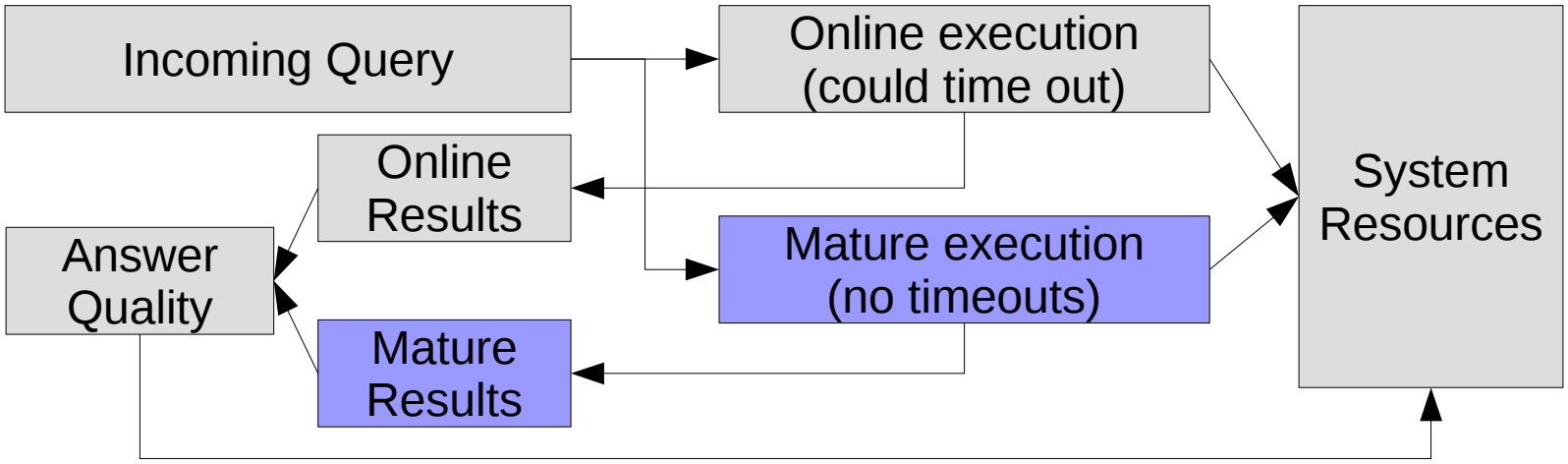
Ubora: A Mature Execution

3. Ubora interferes with Disk access.
4. Redis answers NLP
4. Ubora answers NLP
5. NLP service answers Apache
6. Apache answers Ubora Query
7. Ubora sends Mature Results to Ubora Cache



Answer Quality

- Retrieve Online and Mature Results from Ubora Cache
- Perform Application-Specific Similarity Function to compute Answer Quality



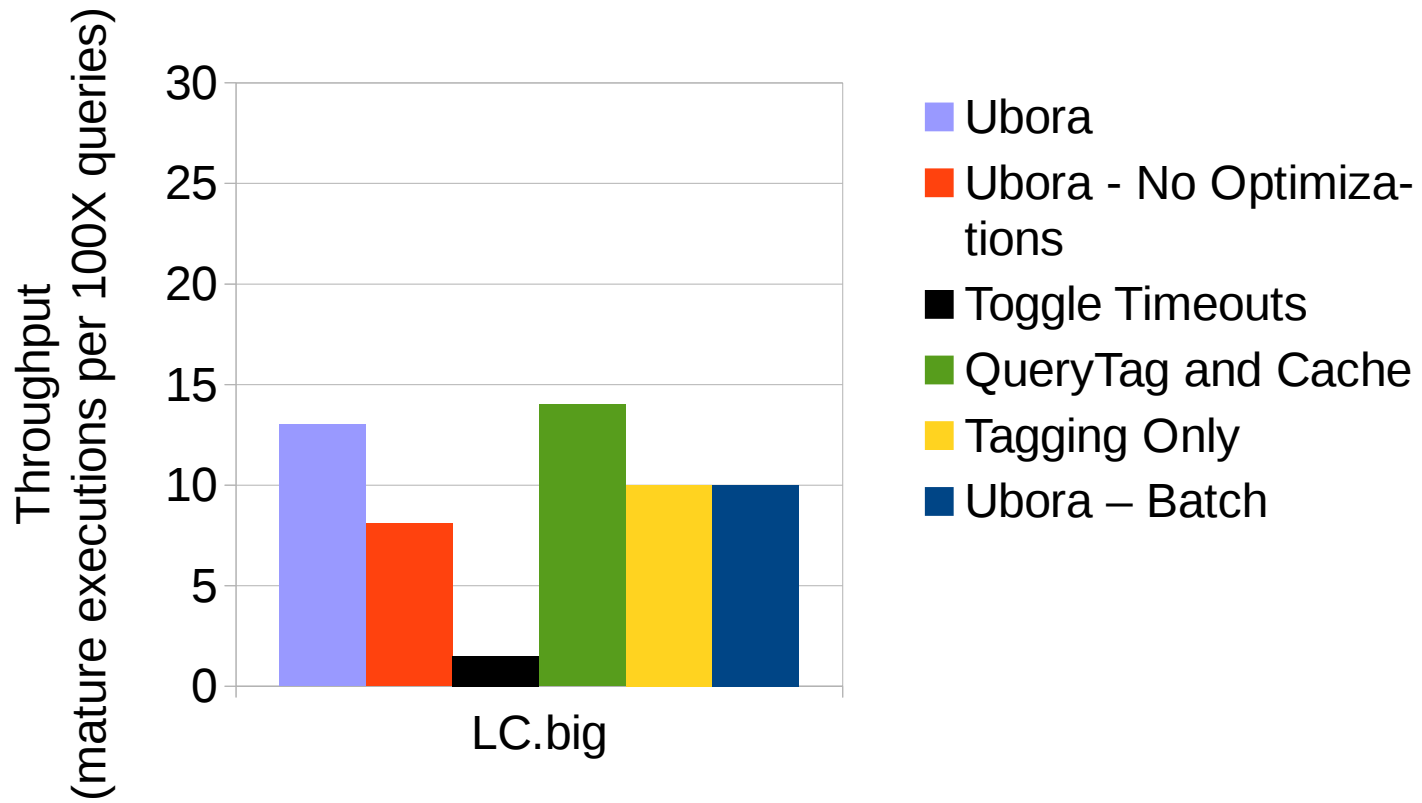
Competing Designs

- Ubora: Full design
- Ubora-Low Samples: Lowers sampling rate, less slowdown
- Ubora-No Optimization: Disables node-local timeouts on Record and Replay to reduce bandwidth
- Query tagging and caching: Ubora at the application level, changes source code to accept per-query timeouts, uses a query cache
- Query tagging: application-level context tracking, no memoization
- Timeout toggling: increases component's global time settings by 4X for mature executions

Workloads

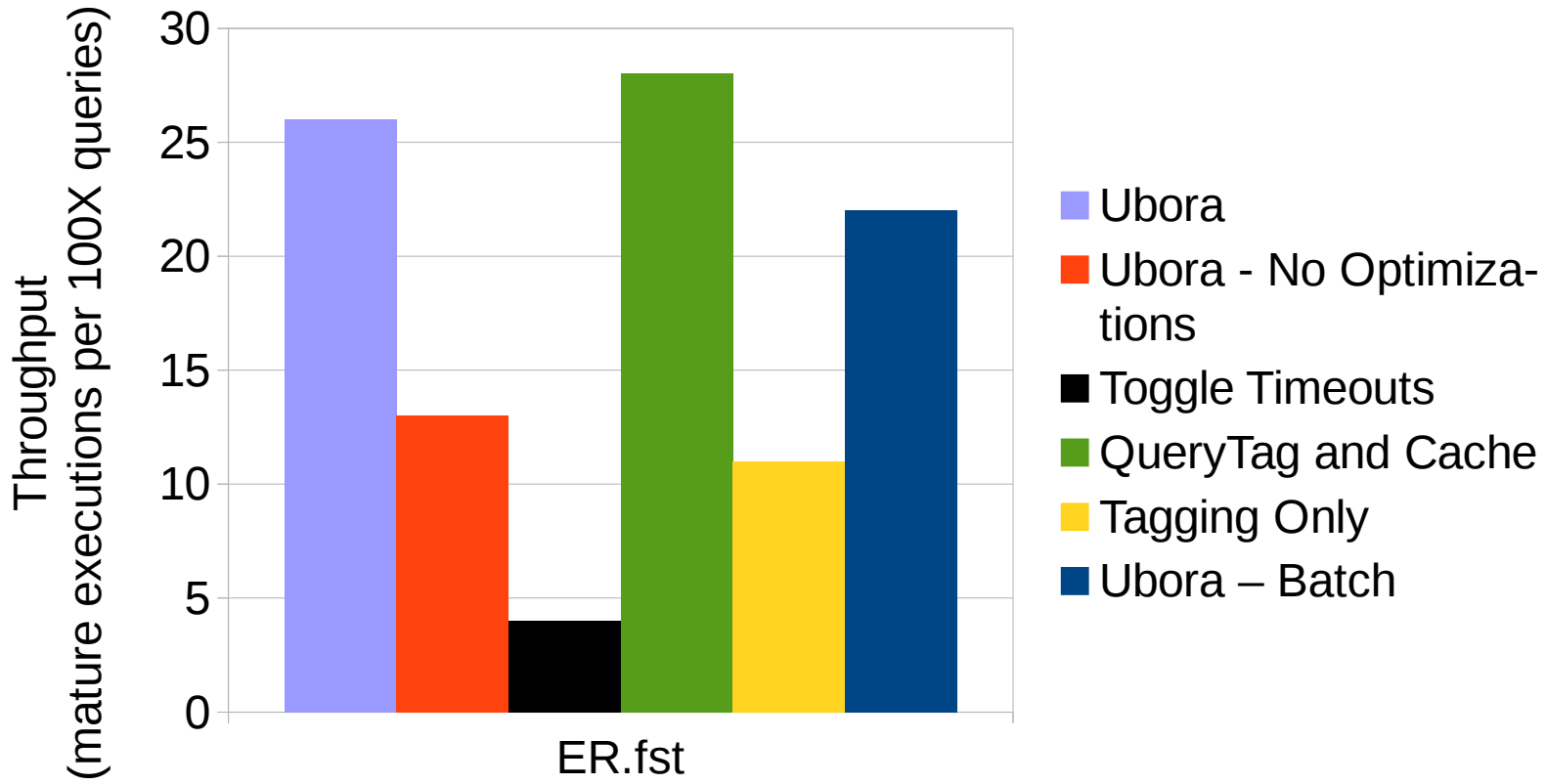
Codename	Platform	Queries	Data	Online (sec)	Mature (sec)
OE.jep	OpenEphyra	General knowledge questions	Wikipedia (just articles)	3	23.43
ER.fst	EasyRec	Movie titles	Netflix similarity data	0.5	0.6
LC.big	Lucene	Google Trends	Wikipedia (with history)	5	23.52
LC.wik	Lucene	Google Trends	Wikipedia (just articles)	3	8.97
LC.news	Lucene	Google Trends	AQUAINT-2 data	1	1.22
YN.bdb	Apache Yarn	Sentiment Analysis	BigBench	178	185

Throughput Results



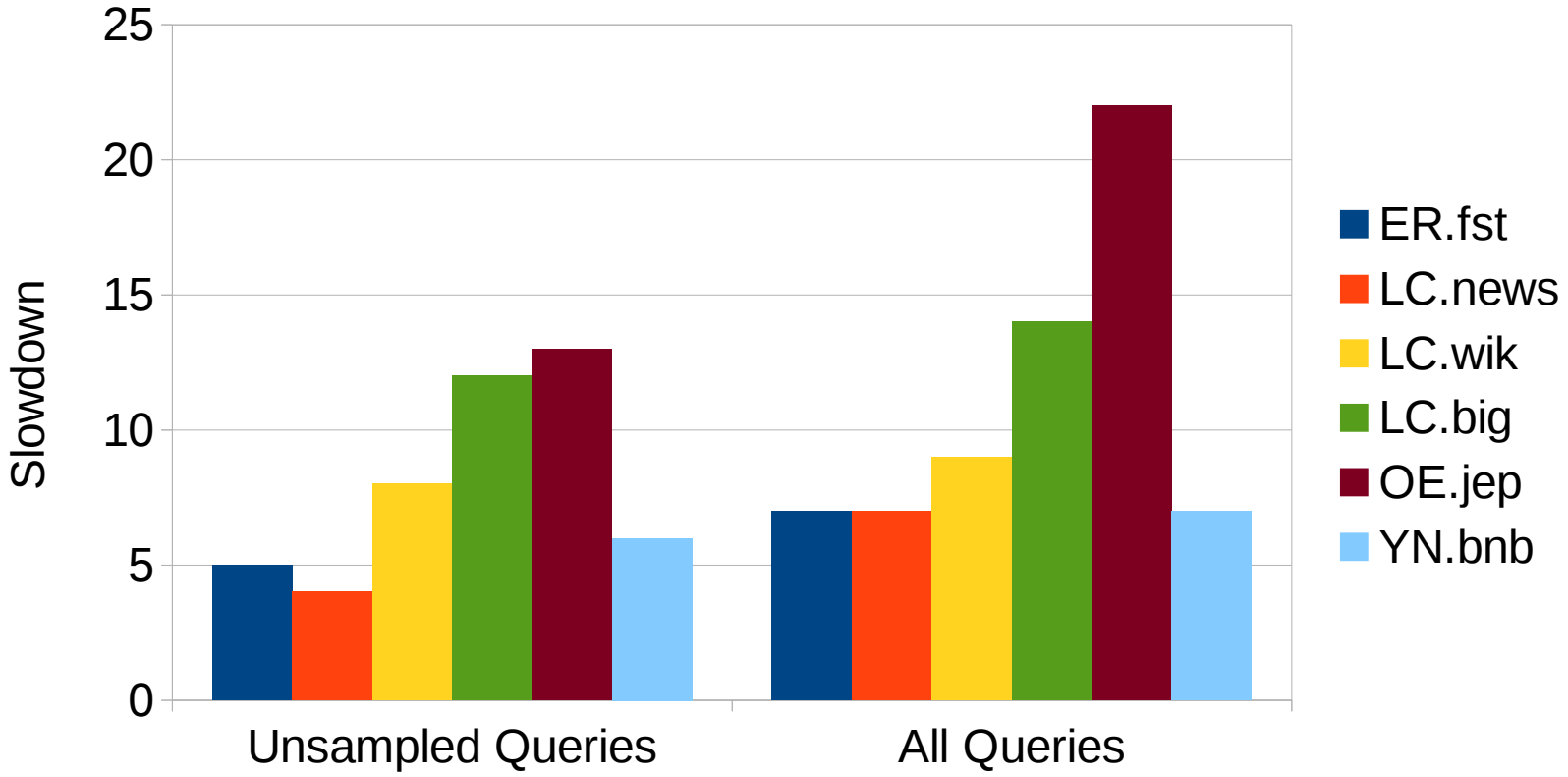
- Ubora performs near throughput of QueryTag and Cache

Throughput Results



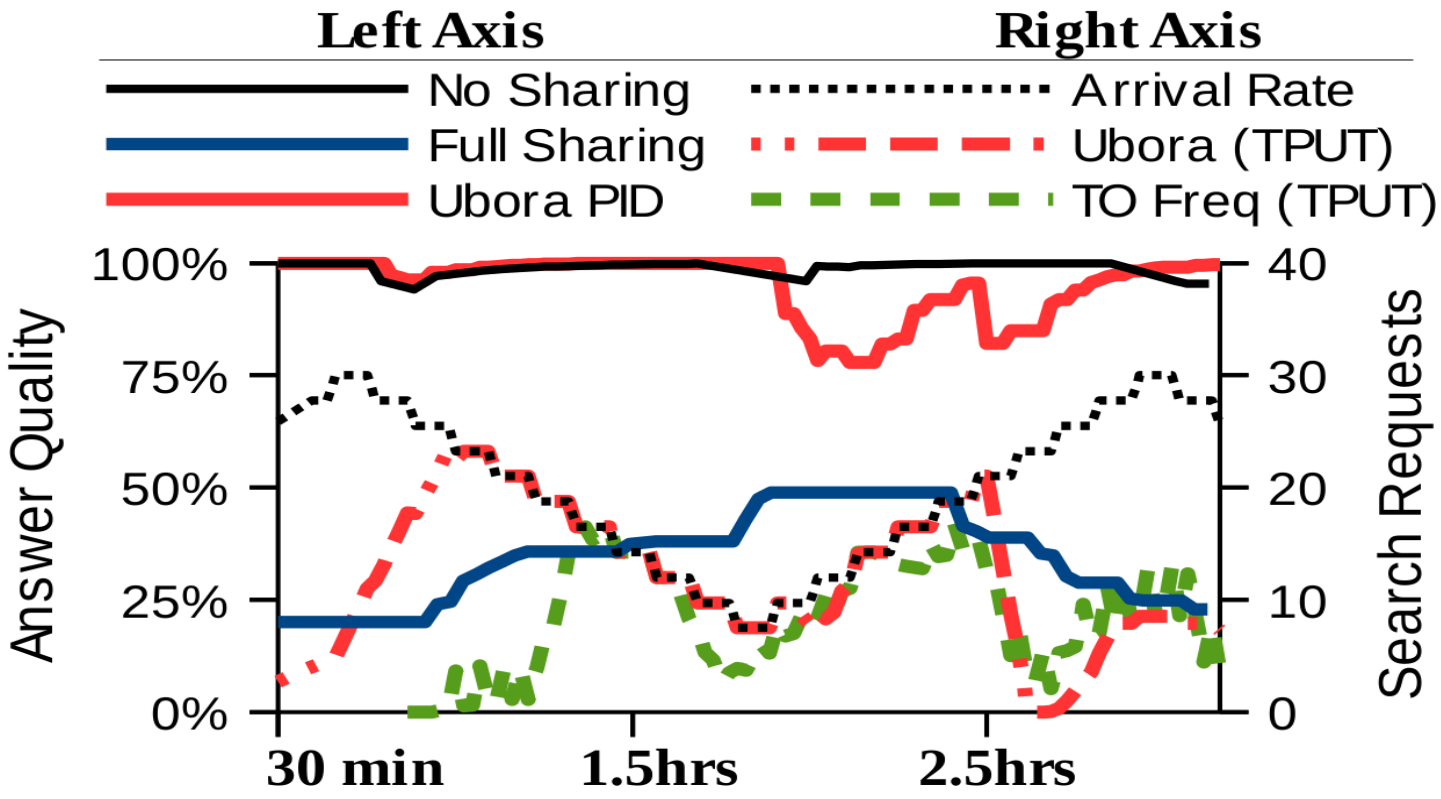
- Ubora performs near throughput of QueryTag and Cache

Slowdown Results



- Unsampled: 7% (AVG), All: 10% (AVG) using Ubora

Load Shedding Results



- Ubora gives better throughput than timeout frequency

Conclusion

Ubora is software that records network traffic and caches this in order to later replay sampled queries.

- Ubora performs at near optimal throughput
- Ubora is a transparent design, requiring no application programming.
- Slowdown on Ubora is less than 10% on average for all queries
- Ubora can be used to manage system resources.