Архитектура поискового кластера

Яндекс
Den Raskovalov
denplusplus@yandex-team.ru
Санкт-Петербург
25.02.2012

О чем это?

Несмотря на то, что автор сведущ лишь в том, как работает поиск Яндекса, слышал о том, как работают другие поисковые машины, он имеет наглость полагать, что приемы, которые накоплены в поисковой индустрии будут полезны любому высоконагруженному сервису обработки данных.

Что умеет Яндекс?

- Зарабатывать деньги (реклама в интернете)
- Делать надежные высоконагруженные сложные сервисы
- Увязывать тысячи источников данных в один продукт
- Решать задачи обработки данных


Яндекс


Яндекс

Requirements

- Indexation
  - ~10^{12} known URL's
  - ~10^{10} indexed documents
  - ~10^{10} searchable documents
  - Real-time indexation
- Real-time query processing
  - 10^{7} user queries per hour in peak
- Development
  - 400+ creative search developers
  - A lot of experiments on full data (pages, links, clicks)
  - Large-scale machine learning
  - Many experiments require full-sized copy of Yandex.Search

Hardware

- Server's cost:
  - Hardware (~$2500 for ~24 computing cores, ~48Gb RAM)
  - Place in data center (~$1000 per year)
  - Electricity for computation and cooling (~$1000 per year)
  - Maintenance (~$300 per year)
- High-end hardware costs too much.
- Low-end hardware needs same electricity and place and maintenance as high-end.
- There is optimum point.
- Now Yandex.Search works on ~10^5 servers.

Distribution of Resources

- Main design principles
  - Sharding (splitting into almost independent parts)
  - Replication (each piece of data is copied several times for performance and failover)
  - Full isolation between development and production (no read/write access for developer to production nodes)

Content System

Content system – system for crawling internet urls, building indexes (inverted index for search, link index, pagerank, ...) and deployment indexes for real-time query processing.

Content System

- Content system phases:
  - Crawling (building a list of URLs)
  - Selection (we can't fetch everything, we need to predict quality before fetching)
  - Fetching document data from Internet
  - Building search representation for documents (inverted indices with lemmatization; binding of external host/url information, link information)
  - Spam detection
  - Duplicates detection
  - Selection (we can't search on everything we fetch)
  - Documents deployment to system of real-time query processing

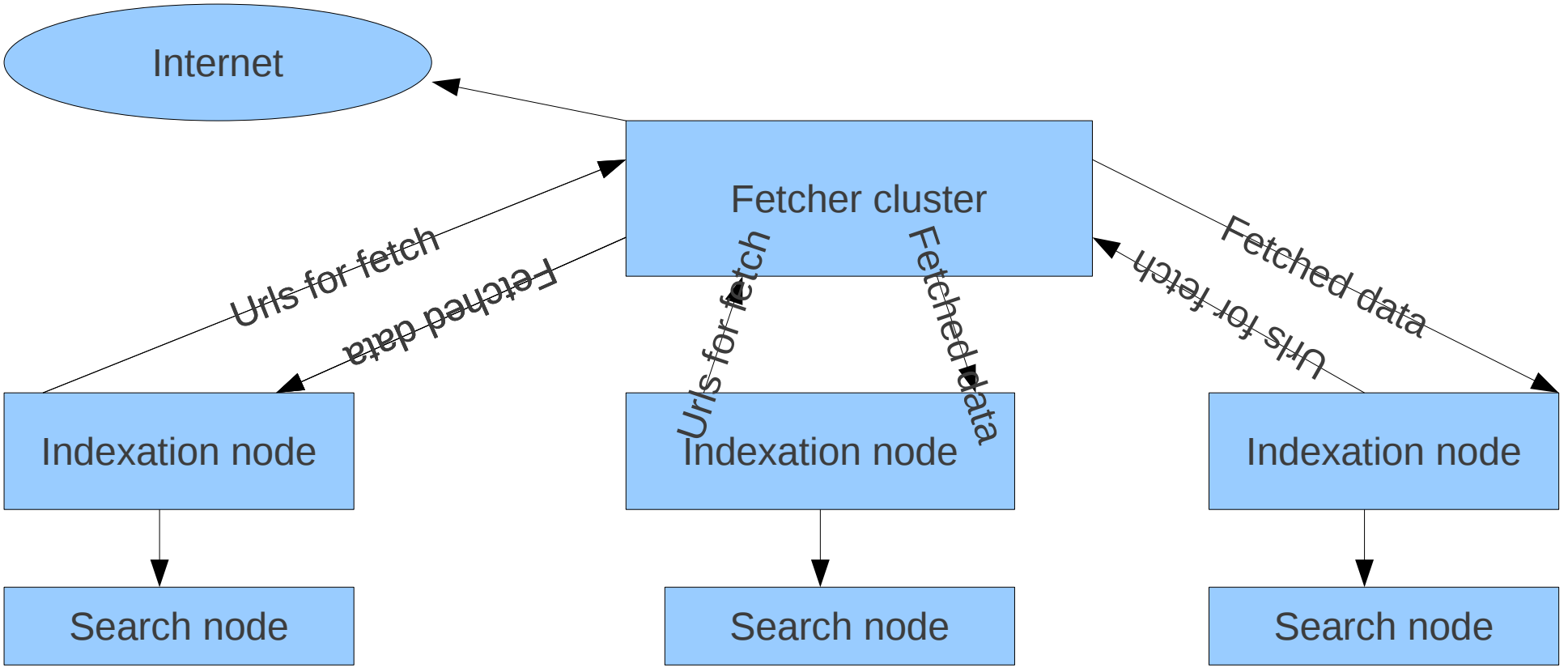
Content System

- In fact we have two content systems now
  - Batch (for processing ~10^{12} URLs)
  - Real-time (for processing ~10^{9} URLs we need for freshness)
- Different update periods inflects architecture
  - Disc-efficient (sort/join) algorithms for batch
  - Almost everything in memory for real-time

Batch Content System

- We split internet in *n* parts based on 32-bit hash of a hostname. We have ~10^4 computers, a computer for an interval of hash values; it crawls “it's” hosts and builds search indexes.
- We represent “knowledge” of internet as relational tables (table of urls, table of links, table of hosts).

Batch Content System



- Once per day each indexation node forms a list of urls for fetch.
- Once per day each indexation node receives fetched data and updates “knowledge” about internet (update process needs only sorting and joins of big tables on disk).
- Once per three days each indexation node push indexes on search nodes.

Batch Content System

- Each indexation node needs “external” data
  - Links from another indexation nodes
    - data passing is asynchronous
    - death of source or target indexation nodes isn't critical (in case of failure data will be received next time)
  - Aggregated clicks on URLs from MapReduce
  - External data about hosts and pages

Real-time Content System

- Main requirement for real-time content system is small time interval between availability of information about document and ability to search on document's content.
- It implies that data can be stored only in RAM. HDD can only be used for eventually consistent backup.
- You have to switch from relational design of data processing to services and message passing algorithms.
- You need big and fast key-value storage in RAM.
- Algorithms with smaller guarantees without consistency invariants (because any network packet can be lose).

Real-time Content System

- We run each set of services on a node:
  - Link database
  - Local pagerank calculator
  - Global dup-detector
  - Anti-spam checker
  - Indexer
- We have library of maintaing key-value storage in memory with eventual backup/restore to/from HDD.
- A lot of network traffic between nodes of real-time content system. Almost no data locality. Nodes of real-time content system need to be in one datacenter.

MapReduce in Yandex.Search

- MapReduce approach is simple, clear data storage combined with data processing, which hides complexity of parallelizing algorithms on several computation nodes
- MapReduce stores set of relational tables. Each table keeps set of records. Each table is physically represented of set of chunks on a HDD (~64Mb). One server (master) knows about all chunks, keeps constraint of replication count >= 3.
- Each record is triple of {key, subKey, value}.

MapReduce in Yandex.Search

- What operations are available:
  - Append set of records to some table
  - Map (iteration over records of some table producing another table, can see only one record)
  - Reduce (iteration over all values with some key)
  - Iteration over table data
- Reduce is implemented with distributed sort.

- Example: calculation of URL click-trought-ratio (CTR) for some query.

MapReduce in Yandex.Search

- We use MR for
  - logs (views, clicks) processing
  - development experiments, which need a lot of data or a lot of CPU
- We have several independent MR-clusters with a total of ~10^3 nodes (some of them are “production”, some of them for data uploading and code execution)

HPC in Search Quality

- We use 'pools' -- conveniently represented or aggregated parts of web data.
- Types of 'pools':
  - LETOR data
  - n-grams frequencies
  - text and HTML samples
  - ....
  - pages with song lyrics from web
  - ....
  - any crazy idea? (you can check it in hours)

HPC in Search Quality

- May be the most important task in search quality is *relevance prediction*. Assume that you have URL-query pairs. We have statistics about url (pagerank, text length), query (frequency, commerciality) and pair {url, query} (text relevance, link relevance); which we call *relevance features.* We want to predict relevance of URL on search query by some feature-based algorithm.
- We need to collect relevance features for thousands queries for millions of documents from thousands computers for training/evaluation of relevance prediction models.

HPC in Search Quality

- Sample of a relevance pool

Query	URL	Label	PageRank	Text Relevance	Link Relevance
yandex	Yandex.ru	Relevant	1	0.5	1
yandex	Hdsjdsfs.com	Irrelevant	0.1	0.5	0.1
wikipedia	Wikipedia.org	Relevant	0.9	1	1
wikipedia	wiki.livejournal.com	Irrelevant	0.01	1	0.01

HPC in Search Quality

- One of the most famous formulas in information retrieval (TF*IDF):

  TextRelevance(Query, Document) = Sum(TextFrequency(Word)/CollectionFrequency(Word)).
- We need to calculate frequency of every word in our document collection for evaluation of TF*IDF text relevance.

HPC in Search Quality

- MatrixNet is the name of our machine learning algorithm.
- MatrixNet can be run on arbitrary number of core processing units.
- It's important because it allows to decrease time for experiment from a week to several hours (if you have 100 computers :)).

HPC in Search Quality

- Many experiments need analysis of difference between Yandex and slightly modified Yandex (beta version).
- Any developer needs the ability to build betas in minutes.
- Requirements:
  - strong system of deployment (you always need to know all parts of system with versions and how you can put it on any node)
  - strong code management (branches, tags, code review)
  - dynamic resource management (you need only several betas at one time)
  - system of automatic search quality evaluation
  - a lot of resources

HPC in Search Quality

- Problems:
  - You need full web data and even more
  - Any computer can be unavailable at any moment
  - You need robust data and code distribution tools
  - You have to execute code near data
  - You need some coordination between different users on development nodes
  - You cannot predict load of development node
- But your progress speed determined by development speed
- Solution: data replication, scheduling of tasks, convenient and robust APIs for developers
- But..

HPC in Search Quality

No final solution.

They are developers. They can eat all resources they can reach.

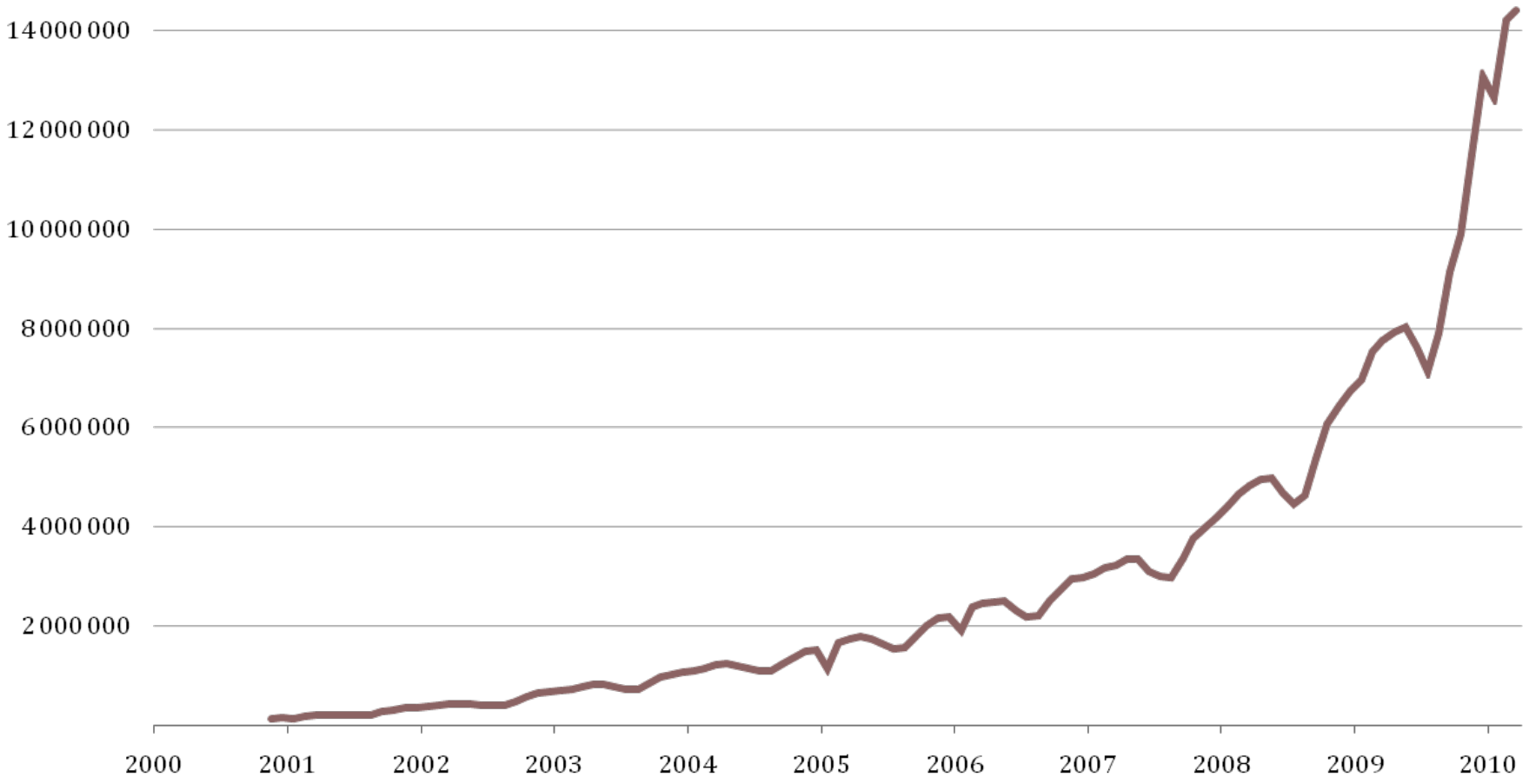
Overview

- Structure of the cluster
- Architectures:
  - Data-centric
    - Batch
    - Real-time
  - Development-centric
    - MapReduce
    - Ad-hoc
  - Real-time query processing

What is this about

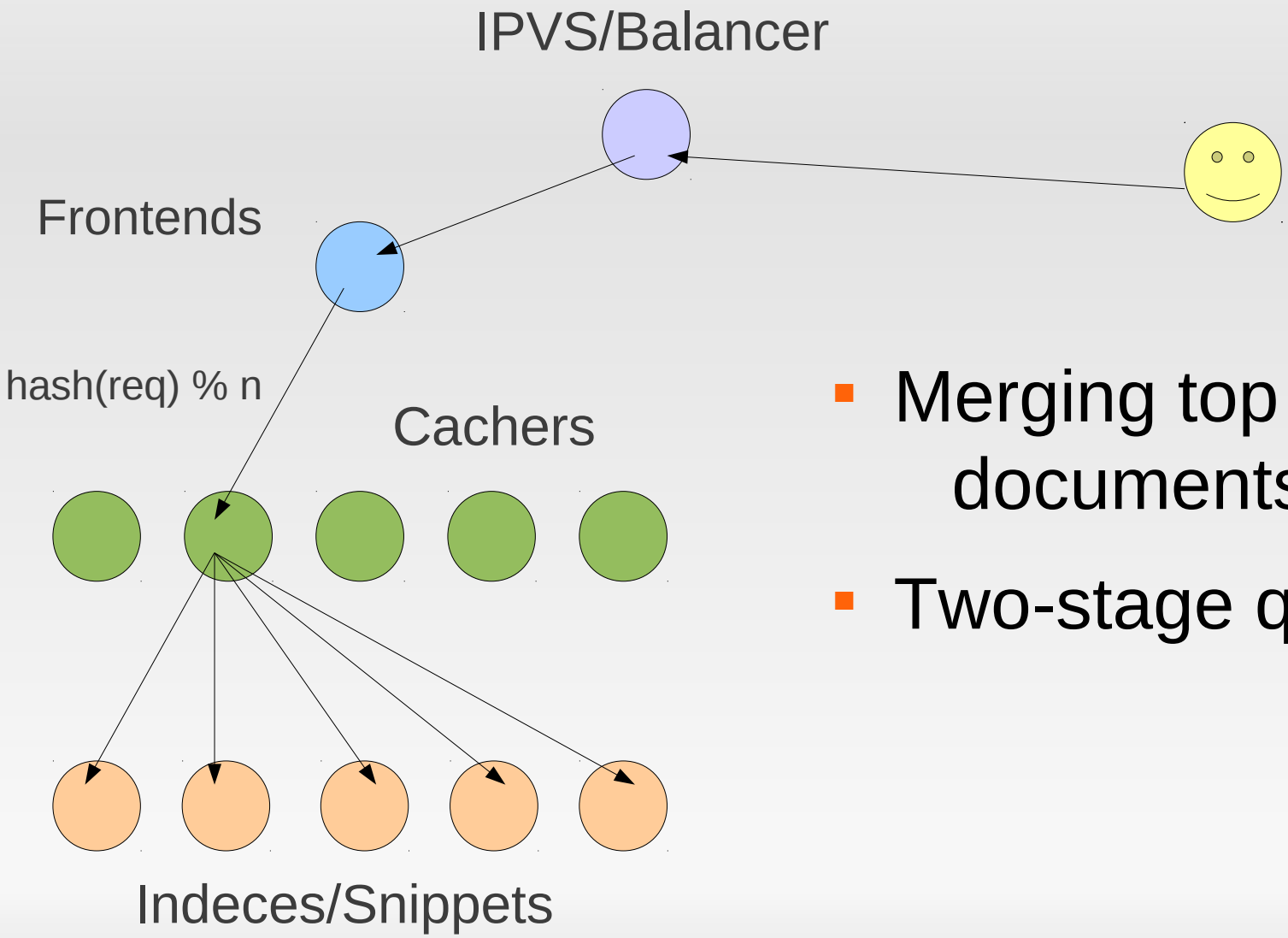
- Computational model of query processing
- Problems and their solutions in terms of this model
- Evolution of Search architecture
- Evolution of Search components

Пользователей в день*



(*) по внутренним данным, в среднем в год; включая экстраполяцию

How does Search work



- Merging top relevant documents
- Two-stage query

Problems

- Connectivity
- Load balancing
  - Static (precalculated)
  - Dynamic (run-time)
- Fault tolerance
- Latency

Network

- Unpredictability of latency
- Bottleneck of the system
- Throughput grows slower than CPU
- Heterogeneous topology
  - 1 datacenter => many datacenters
  - Many datacenters => datacenters in many countries

Load Balancing

- Machines and indexes are different
  - 3 generations of hardware
- Uncontrollable processes at nodes
  - OS decided to clear buffers
  - Monitoring system daemon writes core dump
  - RAID Rebuilding
  - Whatever

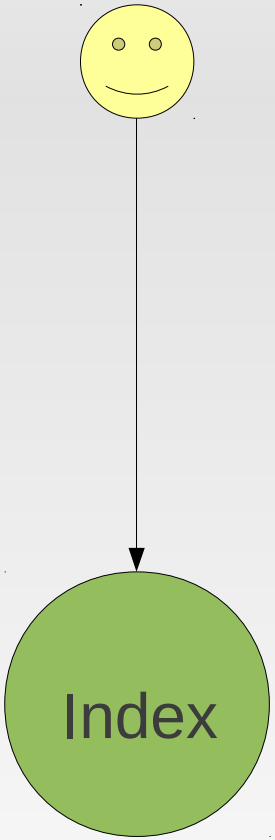
Fault tolerance

- Machines fail
  - 1% of resources is always unavailable
- Packets are lost in the network
- Datacenters get turned off
  - Summer, heat, cooling doesn’t cope with workload
  - Forgot to pay for energy
  - Drunk worker cut the cable
- Cross-country links fail

Latency

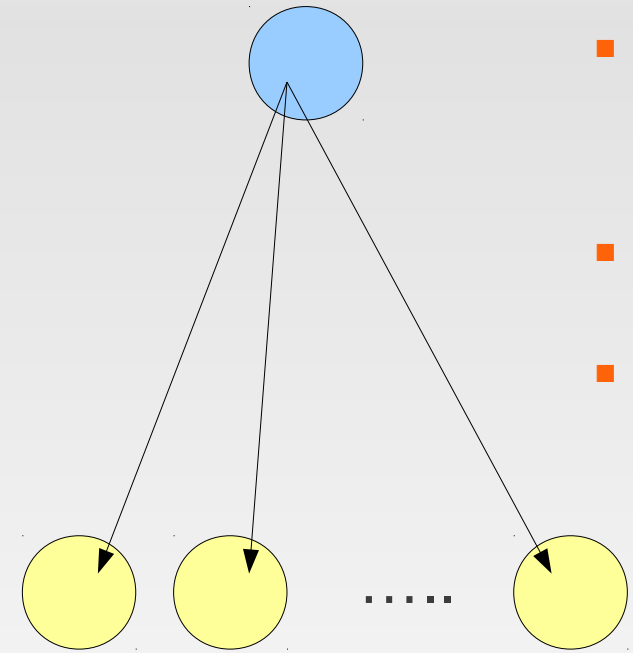
- Different system parts reply with different speed
  - 3 generations of machines
  - Indexes are different
- Random latency
  - Packets are lost in the network
  - Scheduler in the OS doesn’t work
  - Disk is slow (200 S/PS)
- Slow “tail” of queries

15 years ago



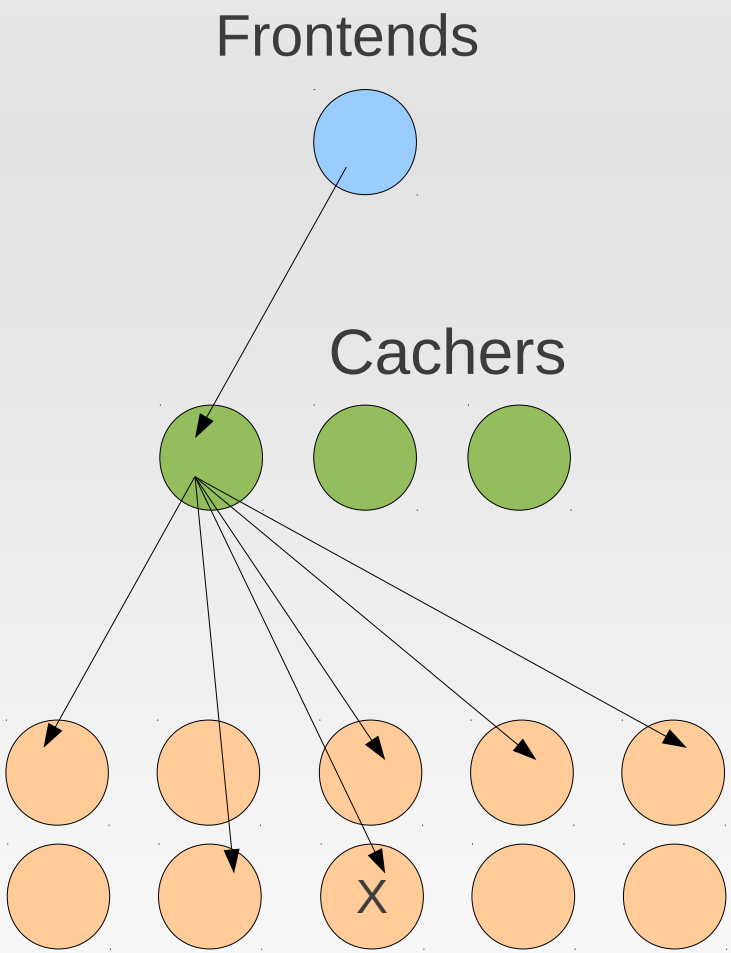
- Everything is on 1 machine
  - Actually, 3, with different DNS names
- Problems are obvious
  - Query volume is growing
  - Base is growing
  - Fault Intolerance

10 years ago



- 1 datacenter, 10 base searchers – the base got distributed
- Can grow query volume and base size
- Got clients and money – delays are unacceptable
  - Machine fails – system doesn't live
  - There are too few machines – subtle things are not noticable yet

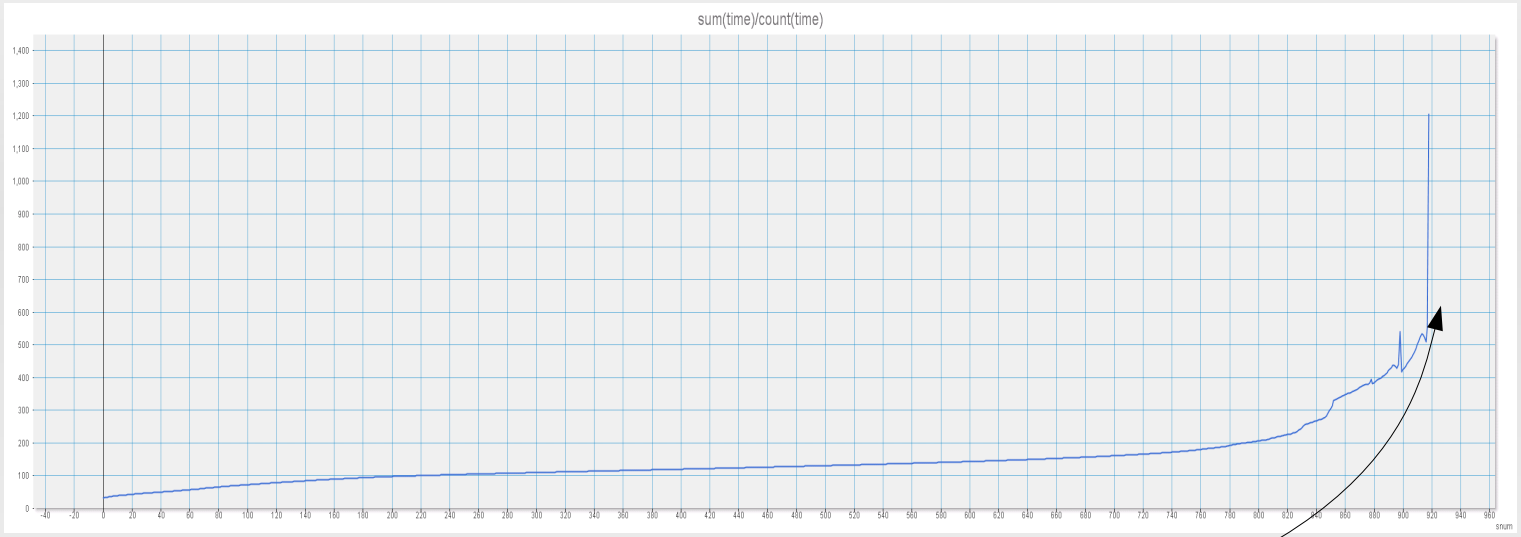
7 years ago



Indeces — 2 replicas

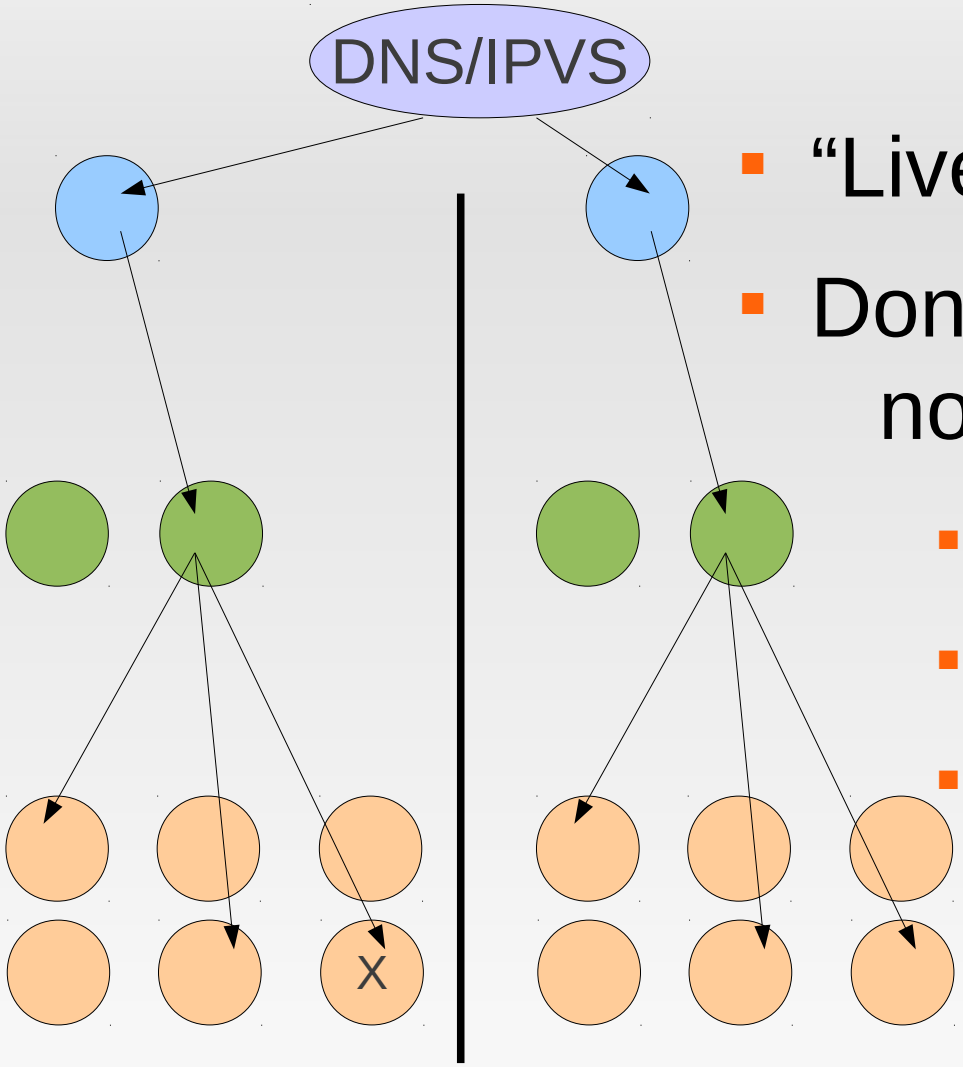
- 300 machines, 1 datacenter, 10 frontends, 20 cachers, 250 base searches
- Problem of machine failures is solved
  - IPVS over frontends
  - Re-Asks at all levels

7 years ago - problems



- Latency
  - Wait for the last source
  - Slow “tail”
- Limit of growth in one datacenter
  - Base is growing
  - Delay because of datacenter failure is unacceptable

4 years ago



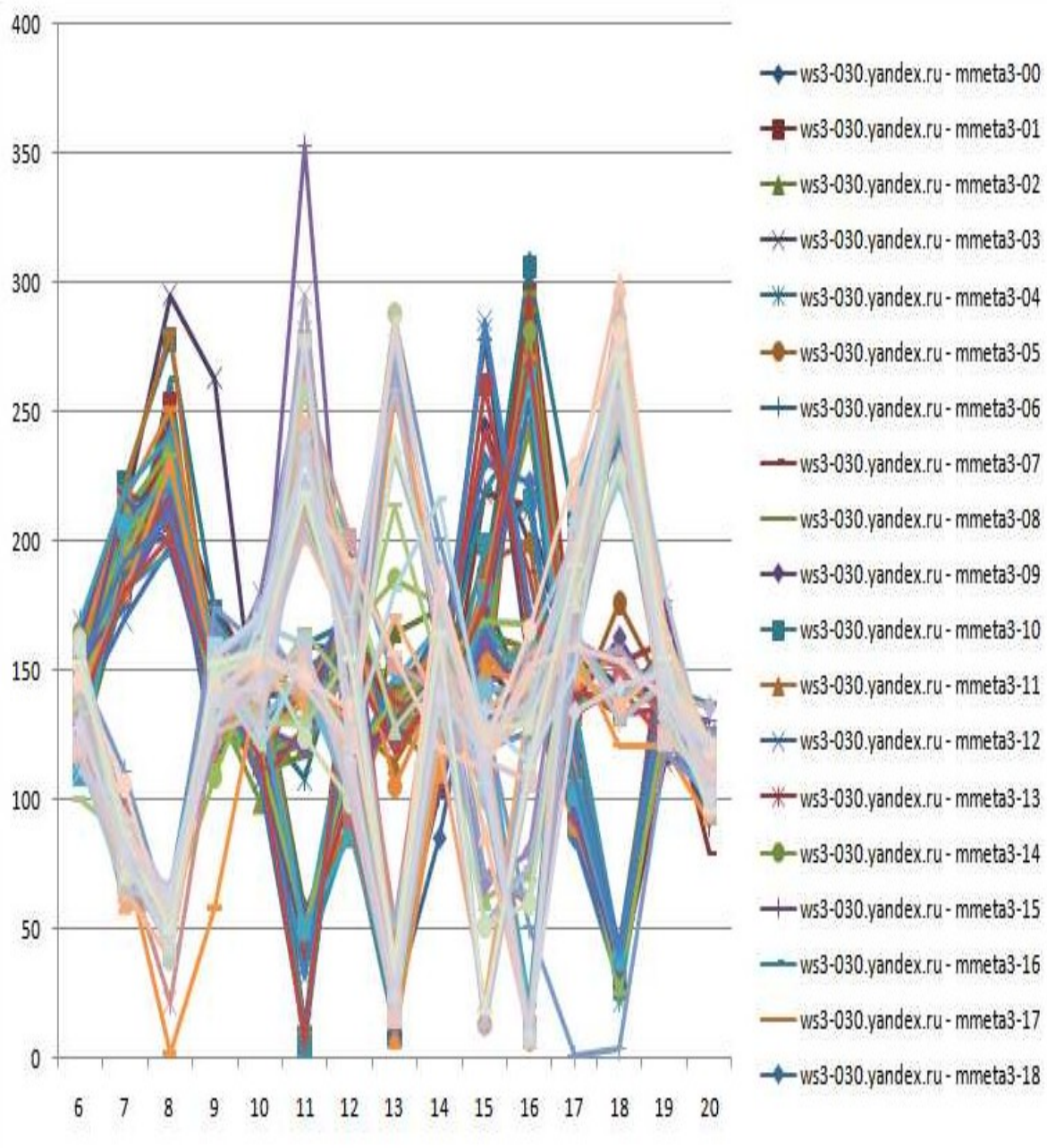
Many Symmetrical Data Centers

- “Live” in several datacenters
- Don’t wait for the slowest nodes
  - Latency is better
  - Partial replies
  - Cache updates after reply to the user

4 years ago - problems

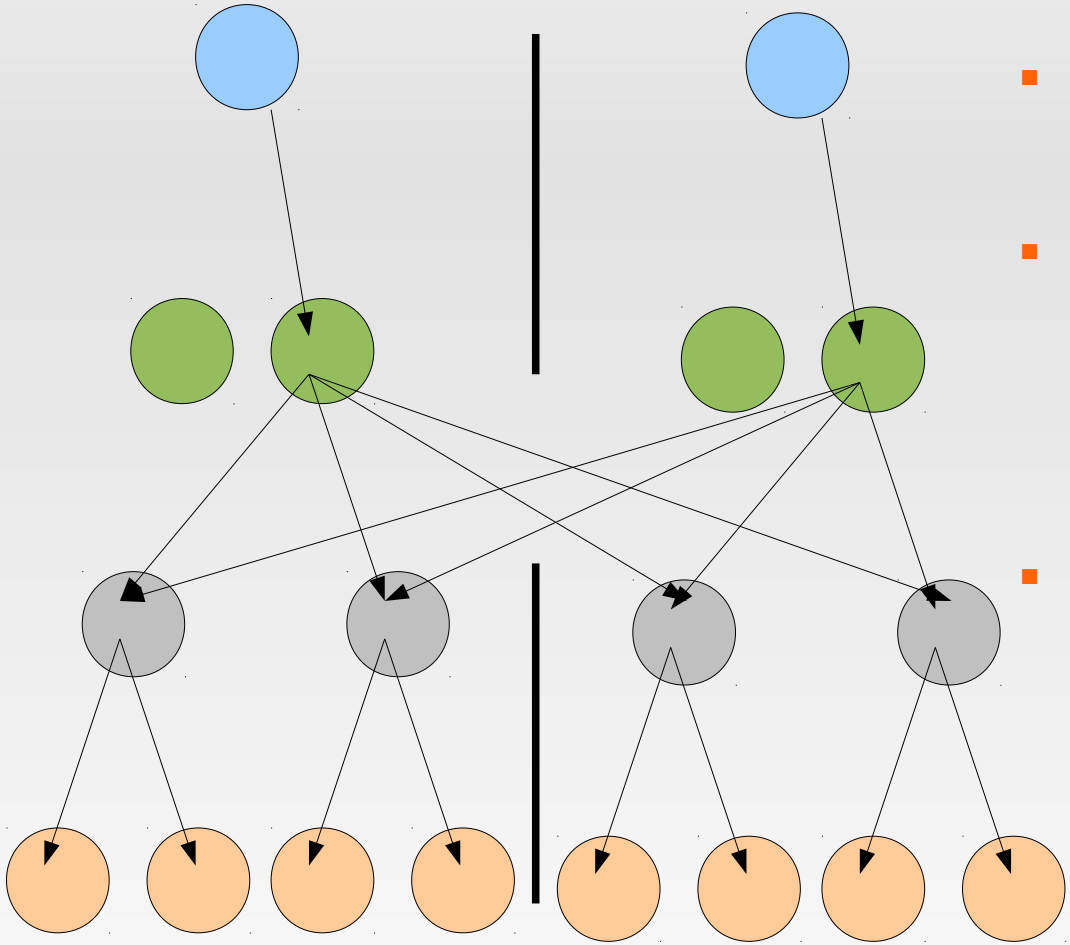
- Datacenters are different, the base is growing, one datacenter can’t hold the whole base
- Need more network even inside the datacenter
- Manual configuration
- “No-reply”s – the scariest word
- Different behavior for different users

Scary Story 2008



- Dynamic Load Balancing
- Hard to implement properly
  - Many sources
  - Fast changes
- Converge guarantee?

What now?



- 10^5 machines, 10^1 datacenters
- Problem of growth in different datacenters solved
  - Integrators – less traffic, can cross DC boundaries
- Cluster structure optimization
  - “Annealing”
  - «cost» → min with fixed RPS and base

Many Datacenters, working as single cluster

Frontier

- New Markets
  - Distributed Data Centers
  - Slow channels
  - Database upload
  - Independent DCs again?
- Latency is “not modern”
  - Ask 2 replicas simultaneously
  - Wait only for the best

Resume

- Succession of architectures
- Linear horizontal scaling
  - By queries
  - By base
- Growth limits are years ahead
- Robustness of computational model

Base Search

- “Conservative” component
  - Stable external interface
  - Complication/optimization of ranking
- Filtering iterator
- Pruning
- Early query termination
- Selection Rank

Cacher/metasearch

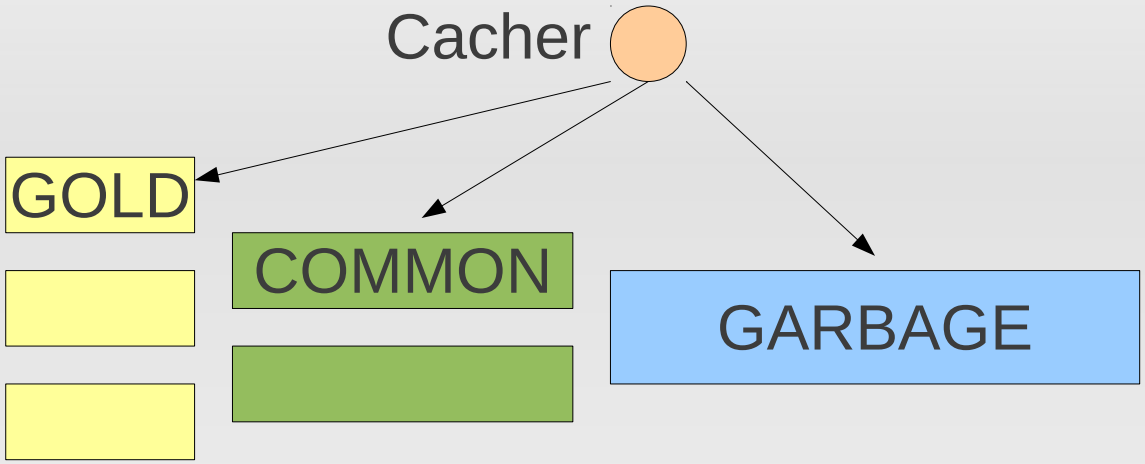
- Synchronous -> asynchronous
  - Message passing
  - Early decision making
- Merge tree
- Rerankings
  - Metasearch statistics
  - Data enrichment

Balancer

- DNS
- IPVS
  - Only inside VLAN
  - Distribution by IP
- HTTP balancer
  - Keepalive
  - Ssl
  - Honest frontend balancing
  - Early robots elimination

Also...

- Tiers
  - Golden
  - Common
  - Garbage



- Index update frequency
  - Main – slow updates (days)
  - Fast – fast updates (hours, minutes)
  - Real-time – ultra-fast updates (seconds)

Further Reading

- Yet Another MapReduce

  http://prezi.com/p4uipwixfd4p/yet-another-mapredu
- Оптимизация алгоритмов ранжирования методами машинного обучения

  http://romip.ru/romip2009/15_yandex.pdf
- Как работают поисковые системы

  http://download.yandex.ru/company/iworld-3.pdf

Thank you.

Questions?