Distributed Architecture for a Peer-to-Peer based Virtual Microscope

Andreas Jaegermann
Departments of Anatomy & Computer Science

Juli 4, 2013
Project Slide Collection

Outline

- Virtual Microscopy
- challenges for distributing WSIs
  - up to 10^5 interactive users
  - gigabyte-sized files
- distribution strategies
- replication model
  - user annotations (dynamic objects)
  - slide images (static objects)

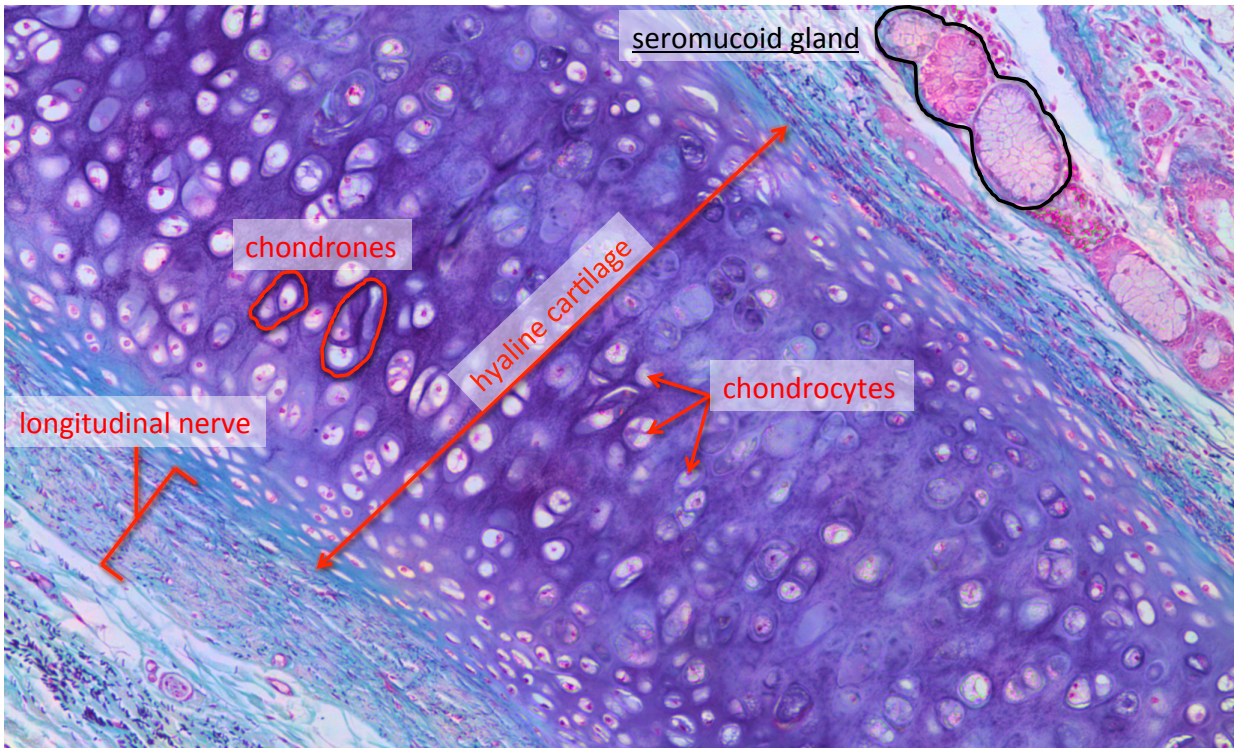
Source: Department of Anatomy, Muenster

Virtual Microscopy

- motivated by the needs of pathologists
- a concept including various aspects
  - data acquisition
  - storage and retrieval
  - visualizing systems

- benefits
  - gathering information from the “tissue”
  - easier Q&A methods
  - over time increasing knowledge base
  - providing “expensive” staining or technique for every student

Virtual Microscopy



Virtual Microscopy

- virtual slides
  - large and mostly proprietary files (~ 6 GB / slide)
  - 150.000 x 260.000 pixel
  - focal planes (~ 40 GB / slide)

- textual (or audio/video) annotations in the slide
  - creation and maintenance (e.g. collections > 5000 slides)

- location-independent learning
  - more than laboratory sized access groups
  - infrastructure
  - application
    - license and library restrictions

DVM: Motivation

- independency from university infrastructure
  - distribution to relieve servers and incorporate the community

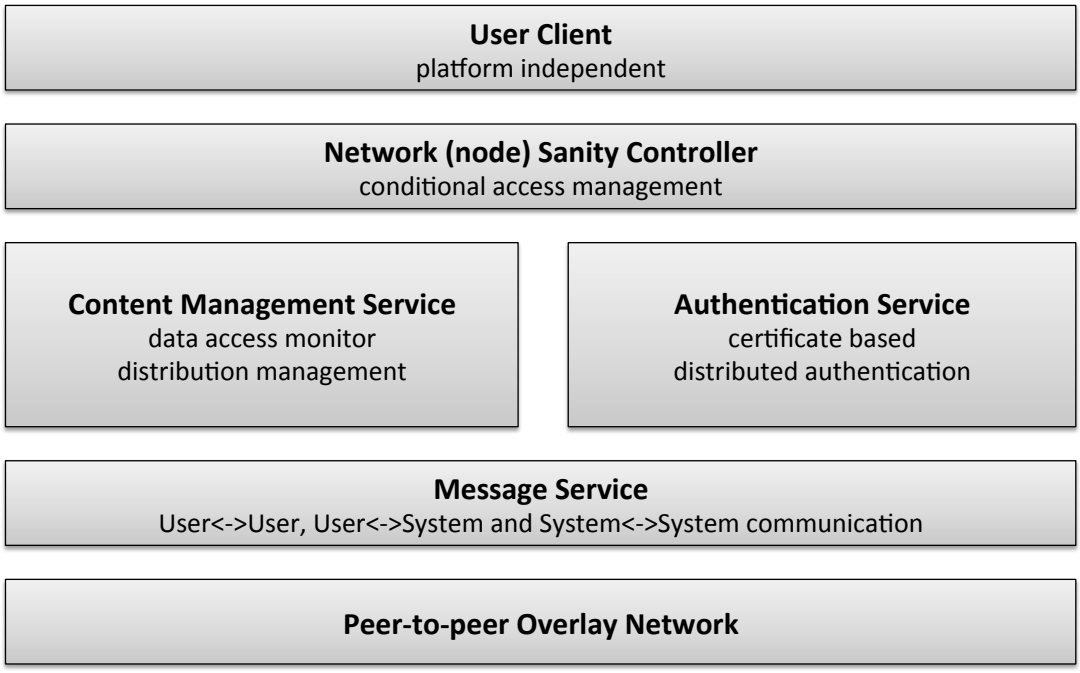
- resource sharing among universities
  - combining slide collections regardless of origin
  - keeping “exam” slides private
  - independent authentication authority

- community maintained knowledge base
  - context-guided learning
  - questions / discussion

DVM: Challenges

- distributed slide files are large
- long range queries to retrieve user content
  - dynamic objects
  - contextually connected among each other
- fast lookup mechanism to locate slides
  - static objects
- replication and load balancing
  - frequently requested parts of slides have to replicated more often
  - user content has to be stored permanently and updated if needed

DVM: Architecture



DVM: Object Definition

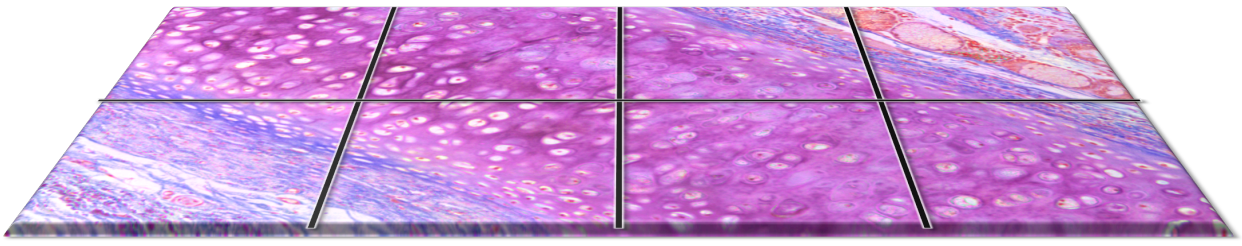
- different groups of dynamic objects
  - instant (e.g. queries)
  - temporary (e.g. position updates)
  - managed (e.g. list of neighbors)
  - durable (e.g. annotations)

- static objects
  - large collections
  - tight request conditions between each other

DVM: Tile Computing

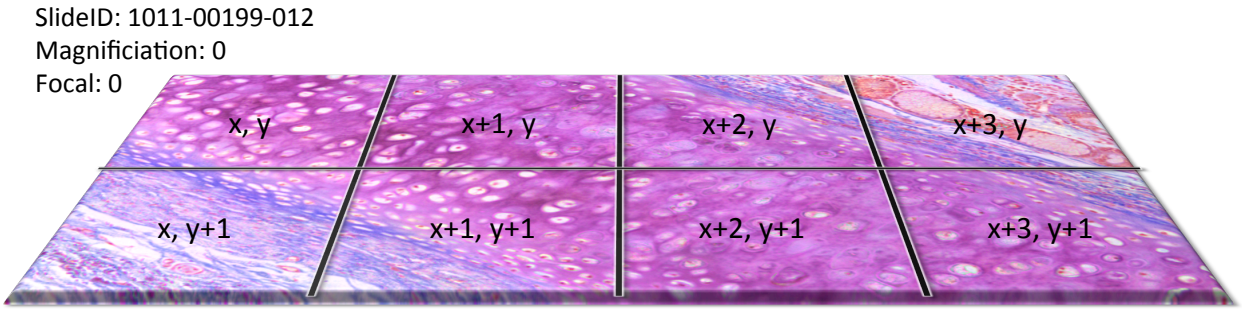
- pre-computing
  - no need for realtime extraction
  - keeping files small
  - retrieving files from multiple sources at once
  - easily more than 10^5 files per layer

```
computeTile(x, y, magLevel, fLayer, width, height)
```



DVM: Tile Computing

- tiles are unique and identified by
  - SlideID
  - Magnification Level
  - Focal Plane
  - Coordinates

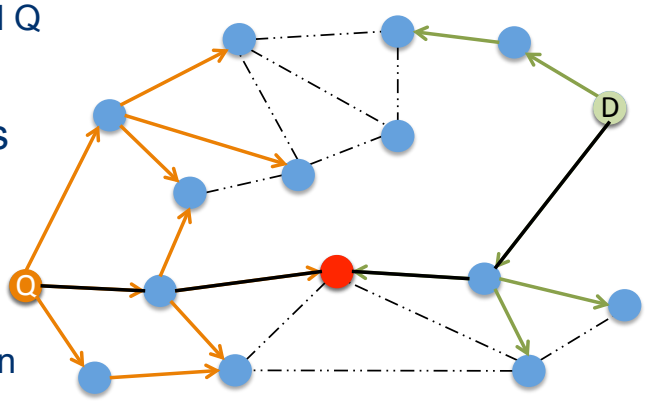


DVM: Distribution Strategies

- requirements
  - efficient management of user-provided content
    - durable replication, updates, removals
  - fast retrievals of easily identifiable tiles
    - replication based on interest to reduce load
- PathFinder (2011)
  - based on BubbleStorm (2007)
    - random graph-based probabilistic rendezvous search
    - highly resilient against catastrophic node failures (up 50%)
    - extended for probabilistic replication in 2008
  - PathFinder adds efficient lookups to exhaustive searches
- DVM adds request-frequency dependent replication model

PathFinder: Object Retrieval

- 2 PRNGs used to create random graph structure
  - determine number of neighbors
  - determine IDs of neighbors
- path from Q to D without network traffic
  - compute tile hash
  - determine neighbors of D and Q
- routing via calculated nodes
- replication model
  - no violation of hash space
  - no further graph fragmentation



DVM: Replication Model

- PathFinder
  - one physical peer controls 1+ virtual nodes
  - inherited replication based on probability
    - replicate objects upon their creation to random neighbors
    - replicate random objects to newly joining neighbors
- DVM
  - extend virtual node over multiple peers
  - replicate objects on node join
  - consider objects from former joins
  - move physical peers between virtual nodes
  - respect request frequency

DVM: Load Balancing for Node JOINs

- control of join location for new node
  - using temporary objects to indicate need for resources

- Example
  1. heavy load on V_5
  2. V_5 pushes temporary object to neighbors
  3. P_5 joins with known node P_4
  4. P_4 redirects to P_3 as V_6 holds a TO
  5. P_3 provides no edge to join but replicates V_5 to P_5

  => load balancing V_5 with P_3 and P_5