VoDKA

An Erlang-based Hierarchical Distributed VoD System

Juan J. Sánchez, José L. Freire, Miguel Barreiro, Victor M. Gulías, Javier Mosquera

LFCIA, University of Coruña

`http://vodka.lfcia.org`

Motivation

Mid 1999 Study with R, local cable provider. Interest in exploring technologies for a later VoD deployment. Market situation indicates VoD strategic in mid-term.

Late 1999 Agreement LFCIA – R to design and develop an experimental VoD system suitable for their fiber and cable network.

Early 2000 EU grant to further support project.

2001 R commitment to large scale deployment

2002 . . . ?

VoD challenges

- Huge storage capacity
- High bandwidth (but relatively few requests)
- Reliable streaming (CBR, VBR...)
- Replacement for TV $\rightarrow$ highly available

- Support for large and tiny configurations
- Adaptability to network topology
- Low cost (how much is people willing to pay?)

Existing VoD systems

Most VoD servers designed for either low-bandwidth on the internet or high-bandwidth corporate LAN

→ **Cable operator customers:** very heterogeneous, links from few hundred kbit/s to many Gbit/sec (56kbps dialups, 150-1200 kbps cable modems, fiber...)

Most VoD servers designed upon a large, powerful, expensive storage server

→ **Cable network:** need for distributed service due to network topology

Existing VoD systems (2)

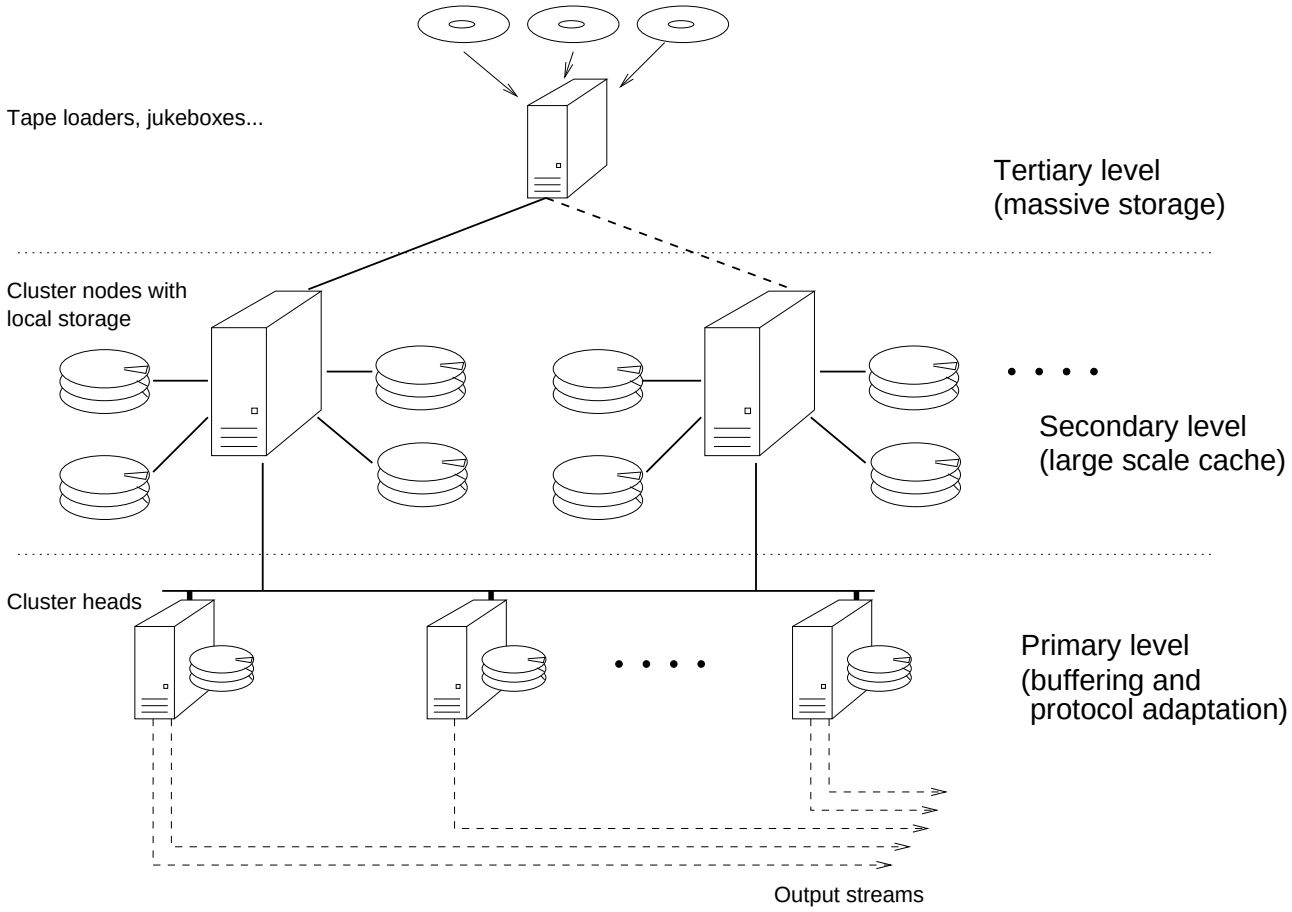
Usually tied to one or few individual video formats, codecs and transports

→ **Cable users:** heterogeneous users (set-top users with existing MPEG decoders, next-generation set-tops, many different computer platforms, kiosks).

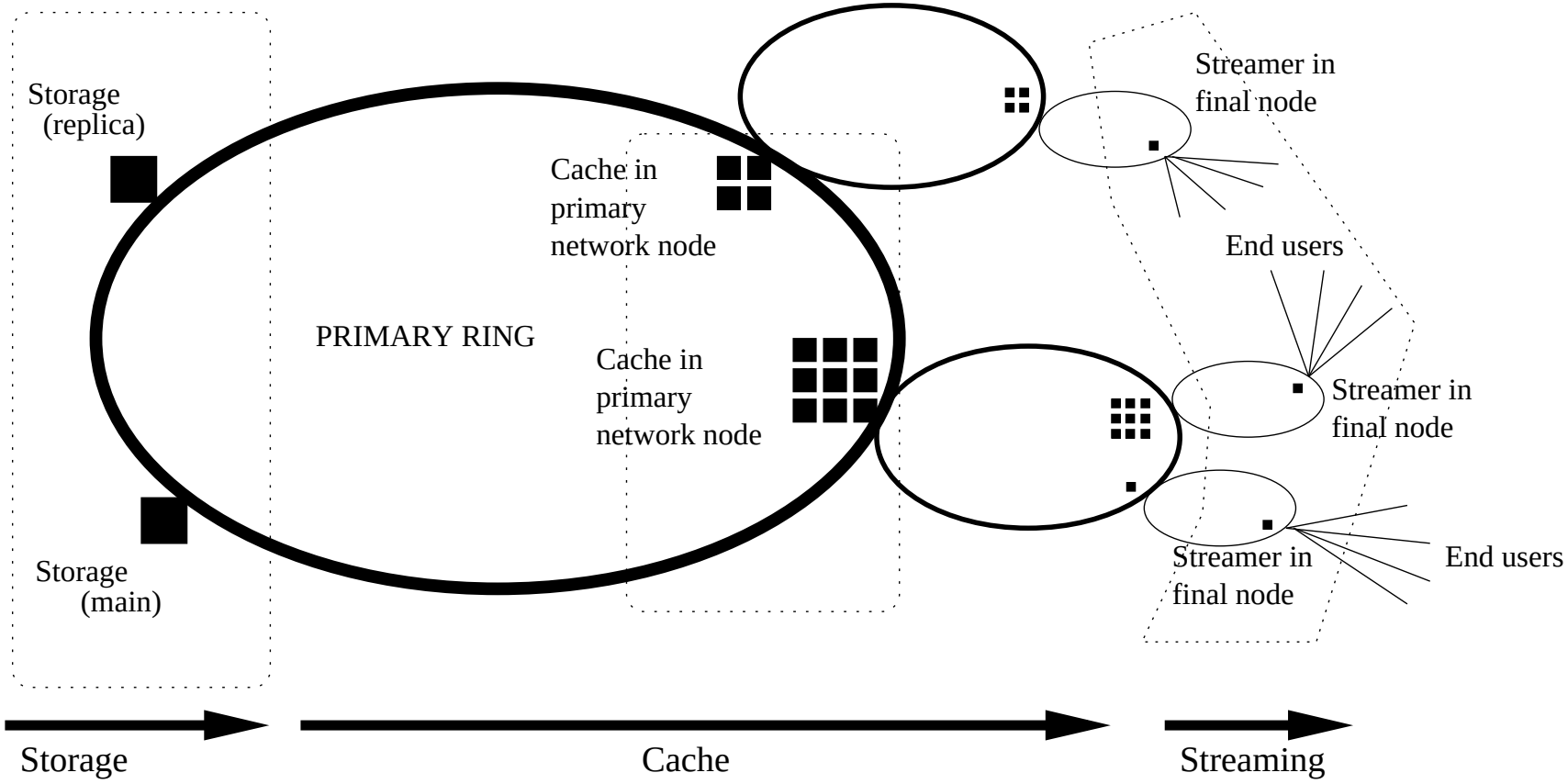
Initial design proposal

- 3-level hierarchy, specialized levels:
  - Massive storage
  - Distributed cache, with replication support
  - Buffering, streaming, protocol adaptation
- Specialized distributed hierarchical storage system. Focus on moving high volume bulk data around with specialized protocols, not on video.
- As flexible as possible

Physical structure



Initial design proposal(2)



Brewing VoDKA

Choosing a development platform:

- Functional background, previous experience with distributed functional languages
- Java experience (thus we tend to avoid it)
- Portability a must
- Small group of developers
- Iterative development cycle
- Need performance, but robustness more important
- Erlang open-sourced

Finally: *Erlang/OTP* (...mostly)

Key technologies

Erlang/OTP: short development time, few errors, thus high productivity. Developer training is critical. Restart coding from scratch? No panic.

Linux: high performance, great portability, robustness, source availability, homogeneous license, fast evolution. Development with cheap hardware, deployment over anything.

Standard protocols: adhere to publicly documented standards.

First lessons learnt

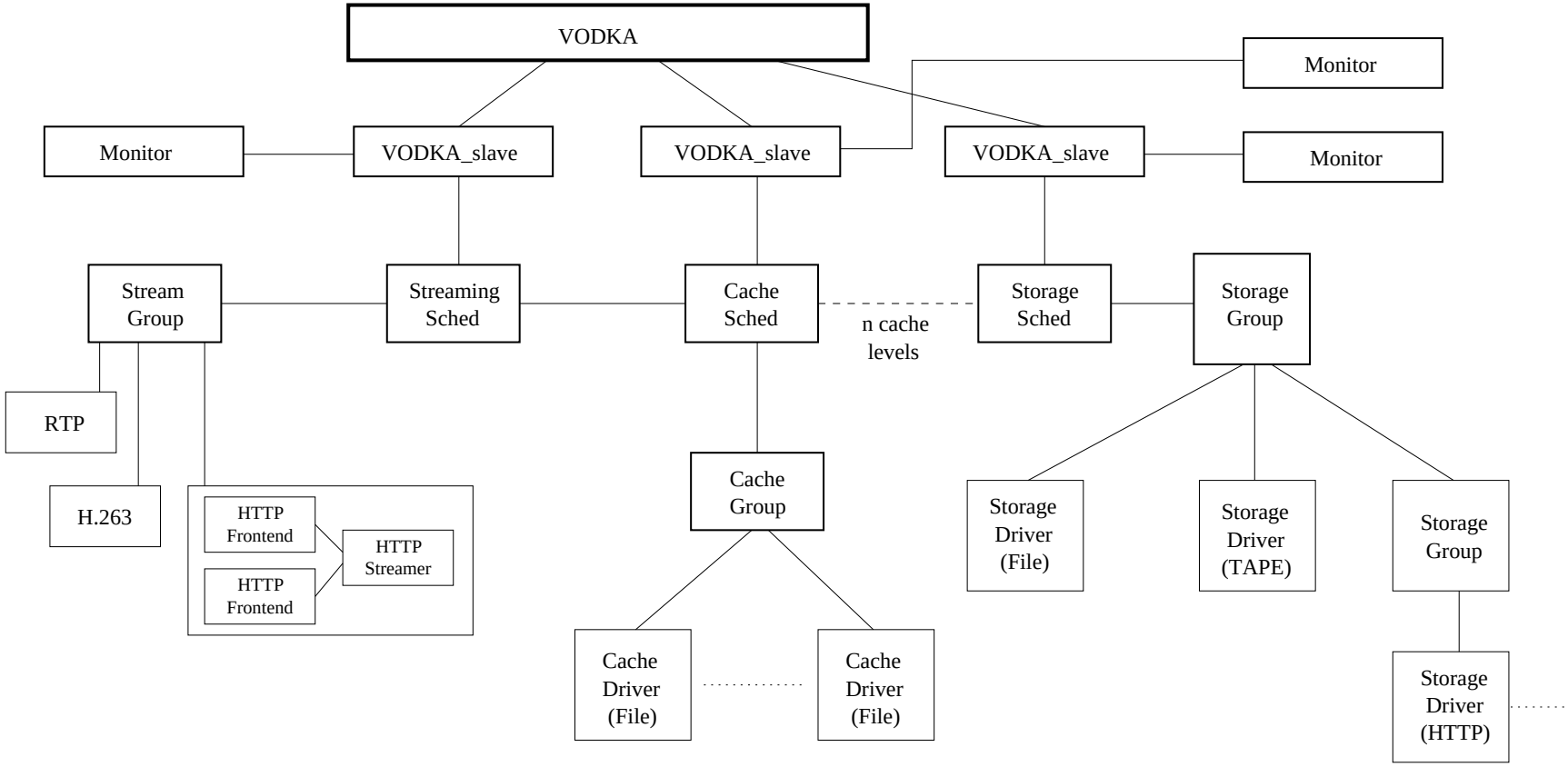
- OTP: high productivity, but steep initial learning curve
- Video formats, codecs, transport protocols nightmare: they change and evolve very rapidly, so don't become dependant on one.
- Ship soon, improve later. Too much time spent on insignificant details (many dumped afterwards). Exposure to real environment will improve product far more (but be responsive!)
- Need for further flexibility. Possibility of small mostly independent systems. Decentralized storage: multiple content providers

Design refinement

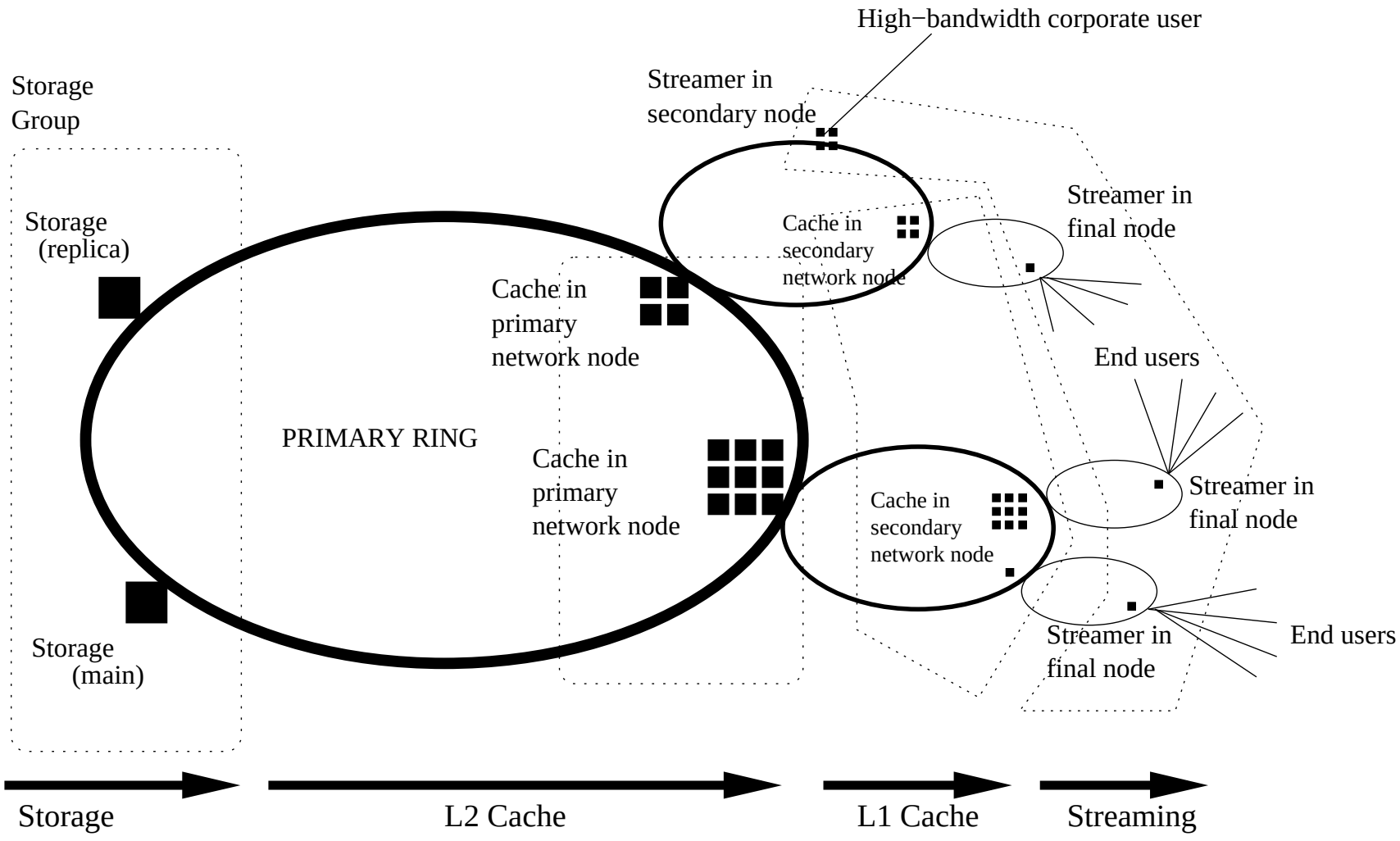
Complete overhaul: attempt to generalize and better suit actual users needs.

- Arbitrary number of levels: small specialized modules (pattern: chain of responsability).
- VoD server cluster as storage subsystem for another server.
- I/O abstraction. Easy to add new transport protocols.
- Reflective servers (introspection).
- Separation of management application and monitoring from the VoD kernel.

VoDKA structure



Physical topology example



Second release changes

- Topology flexibility goal achieved
- Different video object transport protocols implemented
- Run-time selection of transport protocols
- VoDKA explorer
- Web end-user interface (movie catalog, etc)
- More streaming protocols

Ongoing and future work

- Job scheduling, job scheduling, job scheduling... (resource allocation)
- Work on mathematical workload models
- Plenty of room for optimization
- Make it more robust when something goes seriously wrong (hardware failing badly)
- Complex cache behaviour
- Reimplementing from scratch :-)

Final lessons learnt... to date

Erlang :

- The module system is a bit simplistic
- Behaviours should be extensible without recompiling erlang itself
- “Dynamic” type system is a bit frustrating (vs. Caml, Haskell type systems)
- Trained developers’ productivity is great
- Surprisingly good performance. I/O subsystem now 100% Erlang.

Final lessons learnt... to date (2)

OTP :

- Quite extensive (but far from perl or java libraries)
- Quite robust (far more than average perl or java)
- Somewhat steep learning curve.
- Doing individual tasks is well documented, gluing them together is not so well.
- Coherent and rather homogeneous.

Final lessons learnt... to date (3)

Linux :

- “Linux is cheap, Solaris/AIX/IRIX/whatever is more robust but expensive”: **myth**. Linux is robust and not too expensive (but not cheap)
- Linux is extremely portable (Alpha, SPARC, HP-PA, x86, ia64, SH-3, ARM, m68k, s/390) but x86 is more on the safe side.
- Understanding the internals and tweaking things does make a difference
- Very good experience overall.

VoDKA

An Erlang-based Hierarchical Distributed VoD System

Juan J. Sánchez, José L. Freire, Miguel Barreiro, Victor M. Gulías, Javier Mosquera

LFCIA, University of Coruña

`http://vodka.lfcia.org`