Information System for Complex Collaboration Technologies Development

Sergei M. Abramov, Sergej. V. Znamenskij,
Nadezhda S. Zhivchikova, Andrej V. Kotomin, Elena V. Titova

Ailamazyan Pereslavl University

RCDL-2009

Web-based collaborative enterprise for **a long-life project** must keep ***high usability*** for a ***lot of years*** and ***never*** *lose data* sweeping over

☠ hardware failures and migrations ,

☠ OS , server and web - browser software updates ,

☠ changes of objectives (research directions , organization stucture, user demands etc.),

☠ code defects and human mistakes .

Concern on flexical scalability rather then on application logic
$\Rightarrow$ Iterational development , not Rational development .

High reliability, effective access to actual and hidden information, agile user interface are required.

Collaborative interactive development proper support

Guaranteed possibilities required:

- effective system code changes testing in same threads,
- unlimited undo/redo of changes,
- do a lot of various work in different interfaces in a few seconds and watch in real time unexpected results, faults and real conflicts ,
- watch any previously accessable data changes faster or slower.

Time Machine : = *information system with absolute memory , smooth self - developance and asyncronouse real - time access* .

Both ways are dangerous:

non-temporal	Time Machine
Too much of complexity (XML parsers unstable , deadlocks problem , …); **Mission - critical data is unsafe** (may be lost after system crash on unexpected overload and resource exhaustion, programmer or system administrator faults, ...);	**Space - expensive ; innovational risks** (non-relational , non-object-relational, non-postrelational , ...); **compatibility issues ;**

Record structure

Any data element to be permanently saved as a value with key.
Value is an arbitrary string, serialized structure or unique filename.
Key is a concatenation of

- identifying path to element (should not contain '_'),
- char '_' and saving time in seconds (or milliseconds) from 01.01.1970,
- char '_' and signature (= user ID for explicitely modified data),
- access char.

Record key example:
/edu.botik.ru/univer/2009/algebra1/VasyaPetrov/session/mark_1005433290_VAIvanovV

Record access

Access char	The value	Unique name of file with value	Key of other record with the value (link for undo/redo, restructuring and approval)
Testing	**v**	**f**	**l**
Working	**V**	**F**	**L**

In ***working*** environment all testing records completely ignored.
In ***testing*** environment all the records have effect, but saved records never can be marked as working ⇒ **low-cost safe code testing in real system.**

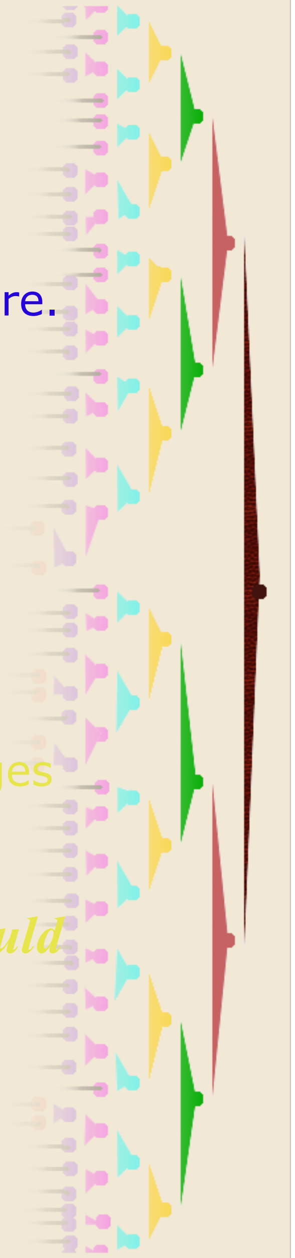
Data access speed

All record keys are lexicographically sorted and accessed via B+Tree structure.
Access to valid on any given (*from the past*) moment record value requires exactly 1 read of B+Tree record (if no undo/redo applied).

OS read caching should work fine:

- semantically close data is more likely to have close identifying paths
  ⇒ have great chance to fit the same memory pages to be cached.
- If commonly prefixed record set is out of need, then the memory pages will not be read.

Data indexes are based on ordinary records with specific prefixes and should also be history-safe and fast.



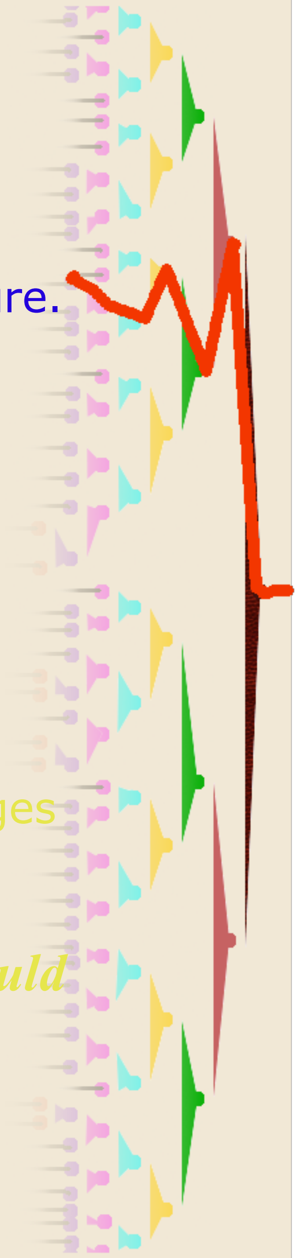
Data access speed

All record keys are lexicographically sorted and accessed via B+Tree structure. Access to valid on any given (*from the past*) moment record value requires exactly 1 read of B+Tree record (if no undo/redo applied).

OS read caching should work fine:

- semantically close data is more likely to have close identifying paths
  ⇒ have great chance to fit the same memory pages to be cached.
- If commonly prefixed record set is out of need, then the memory pages will not be read.

Data indexes are based on ordinary records with specific prefixes and should also be history-safe and fast.



Data access speed

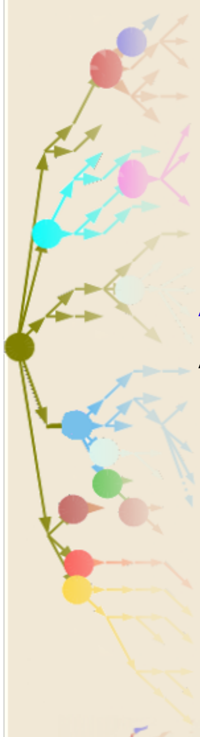
All record keys are lexicographically sorted and accessed via B+Tree structure.
Access to valid on any given (*from the past*) moment record value requires exactly 1 read of B+Tree record (if no undo/redo applied).

OS read caching should work fine:

- semantically close data is more likely to have close identifying paths
  ⇒ have great chance to fit the same memory pages to be cached.
- If commonly prefixed record set is out of need, then the memory pages will not be read.

Data indexes are based on ordinary records with specific prefixes and should also be history-safe and fast.

⇧ Information tree of key prefixes



Data access speed

All record keys are lexicographically sorted and accessed via B+Tree structure.
Access to valid on any given (*from the past*) moment record value requires exactly 1 read of B+Tree record (if no undo/redo applied).

OS read caching should work fine:

- semantically close data is more likely to have close identifying paths
  ⇒ have great chance to fit the same memory pages to be cached.
- If commonly prefixed record set is out of need, then the memory pages will not be read.

Data indexes are based on ordinary records with specific prefixes and should also be history-safe and fast.

⇧ Index tree

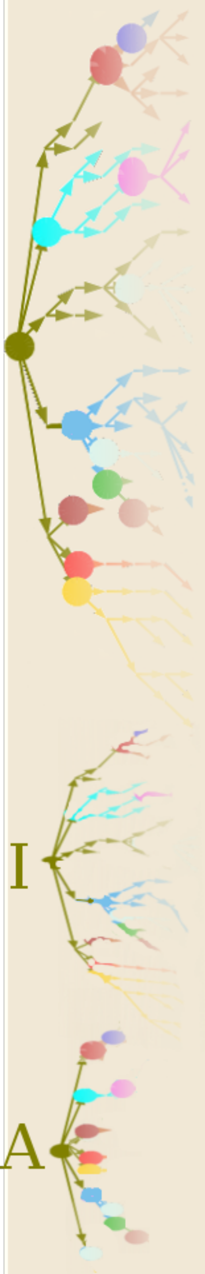
Data access speed

All record keys are lexicographically sorted and accessed via B+Tree structure.
Access to valid on any given (*from the past*) moment record value requires exactly 1 read of B+Tree record (if no undo/redo applied).

OS read caching should work fine:

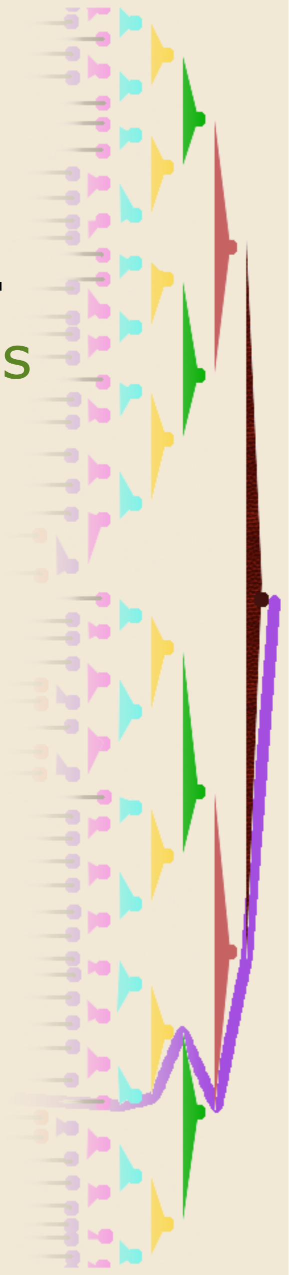
- semantically close data is more likely to have close identifying paths
  ⇒ have great chance to fit the same memory pages to be cached.
- If commonly prefixed record set is out of need, then the memory pages will not be readen.

Data indexes are based on ordinary records with specific prefixes and should also be history-safe and fast.



Access control (low level)

- Identifying paths form the system information tree.
- Any node may be selected to be control (bold circles on picture).
- Control node path with special prefix forms key of special record.
- Access restrictions are serialised to the value of control record.
- Such condition acts on a maximal part of branch grows from control node until other control nodes (*data context*).
- Exactly 1 read of B+Tree record returns access conditions.



Server query cycle

- activise or load appropriate ***executable code versions***;
- check server load and perform a reasonable part of
  - fast respond to browser based on a ready data from database.
  - current *data context* executions,
  - indexing,
  - database optimisation, …,

  preparing (partial also) results for database;
- ***send all unsaved data to database***;

Results of *data context* partial executions must only rely on sly saved data, not on currently available data:

redundancy makes no problem , concurency do.

Smart browser

- periodically (0.5 sec. ... 10 min.) send to server ***asynchronous update requests*** (including unsaved data changes if any);
- ***tune periodicity*** (e.g. to 0.3 sec. after editing, 2 sec. after link hiting, 20 sec. after short pause, 10 min. after long pause);
- if user change ***form field***, remember ***changes as unsaved data***;
- on respond
  - ***update screen*** fragments if necessary,
  - remember ***succesfully saved*** user data not to send more until other changes occure.

Running code life cycle

- create/edit/save for personal testing with personal code selections
- test in personal testing environment
- test with current beta testing environment
- propose for beta-testing environment
- approve for beta-testing environment
- approve beta-testing environment for production use
- upgrade system

Unlimited delete/undo/redo and ***select/approve/abandon***

Step 1: Software selection

General requirements: high stability, high performance,unlimited flexibility, open source, well-documented, well-tested, well-supported, convenient API.

Apache2+ModPerl2: expressive featured scripting language, able to load on-the-fly code from database, internaly designed not to load same code twice.

TokioCabinet: convenient perl interface, partial key matching support, concurent data store support.

Step 2: Database and code loader

General requirements: minimal clear code, high performance, unlimited flexibility, free version combining and selection, testing in working system.

If (b.01 tested with c.02 and d.03 tested with c.01) then possibility to test b.01 with d.03 with both versions of c.

Step 3: Integrated Programmer interface

General requirements:

- Version descriptions, documentation, testing errors and user requests are on the ***version web-page***.
- ***Safe code testing*** and system updates without server reload.
- Possibility to ***revert any changes*** without loose history.
- ***Time machine*** control panel.

Step 4: Project and conferencing support
Step 5: Highly reliabile distributed system
Step 6???

Information System for Complex Collaboration Technologies Development

Sergei M. Abramov, Sergej. V. Znamenskij,
Nadezhda S. Zhivchikova, Andrej N. Kotomin, Elena V. Titova

Ailamazyan Pereslavl University

RCDL-2009