`vos each`: *Query the VLDB in Style*

Nathaniel Wesley Filardo

August 19, 2015

Outline

Motivation

Ever written one of these?

```
vos listvldb ... | grep ... | ... | xargs ...
vos listvol ... | while read ...; do ...; done
```

Does this leave a warm fuzzy feeling?

Motivation

- `vos backupsys` exists for that special case. But I want to do something else. I want to *release* volumes, not back them up.

Motivation

- `vos backupsys` exists for that special case. But I want to do something else. I want to *release* volumes, not back them up.
- So I should add a `vos releasesys`. And then later another `vos` subcommand... and... surely a better way!

Motivation

`vos each`: CLI for generic VLDB queries.

Describe all RO volumes on a server's partition:

```
$ vos each -ro -partition vicepa \
  -server afs0.example.com
```

Output: "`-volume testvol -server 93.184.216.34 -partition /vicepa -id 536872491`".

Motivation

`vos each`: CLI for generic VLDB queries.

List of all unlocked "old replica" volumes:

```
$ vos each -unlocked -ro -rodontfl \
  -format '%v (%n) on %s:%p'
```

```
testvol (536872491) on 93.184.216.34:/vicepa
```

Motivation

`vos each`: CLI for generic VLDB queries.

Parallelize a `vos backupsys` of some volumes?

```
$ vos each -rw -server afs0.example.com \
 -iregex 'chicago$' -eprefix 'temp' \
 -format 'vos backup %v' \
 | xargs -P 3 -I {} sh -c {}
```

```
vos backup admins.chicago
```

Motivation

- `vos each` is (I think) well documented in the manual.
  - $\sim$ 1000 source lines in diff; $\sim$ 400 lines in POD doc.
- Still needs reviewers!
  - If this makes you happier than

    ```
    vos listvol | grep ... | ...
    ```

    mind having a look for me at
    `http://gerrit.openafs.org/#change,10966` ?

Using `vos each`*: Queries*

`vos each` encapsulates a simple idea:

> Find all volumes in an AFS cell that match some critera.

Two parts:

- What kind of criteria?
- And once we've found them, . . . ?

Using `vos each`*: Queries*

`vos each` can make conjunctive queries over...

- Volume group name: `-{i,e}{glob,regex,prefix}`
  - Inclusive or exclusive match
  - glob, (platform) regex, or literal prefix
- `-server` and `-partition`
- Volume type: `-{ro,rw,bk,clone}`
- Existence of a type in a group: `-e{ro,rw,bk}`
- Replication status: `-rodontfl`, `-newrepfl`
- `-locked` or `-unlocked`

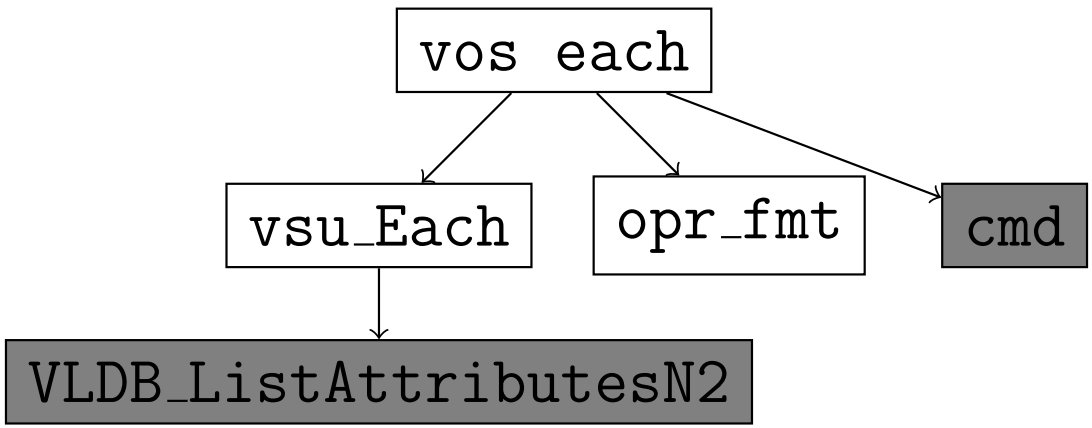
Some more, mostly for debugging the VLDB; all documented!

Using `vos each`*: Answers*

- For each match, `vos each` invokes a `sprintf`-alike.
- Format string given as `-format` on command line.
- Available escapes are everything I could think of, including:
  - `%v`: Volume group name
  - `%n`: Volume identifier (numeric)
  - `%s`: Server in dotted quad form
  - `%p`: Partiton as "`/vicepX`".
  - `%t`: Type ("`RW`", "`RO`", "`Backup`", "`Clone`")
  - Numeric volume IDs in the group:
    - `%B` backup, `%C` clone, `%R` RO, `%W` RW

  And others (mostly of interest while debugging).

Components



Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:

```
struct nvldbentry *ent; // New VLDB Entry
unsigned int eix;       // Entry index
VolumeId volid;         // Volume ID
unsigned int svix; // Server index
// ... some other useful things, too
```

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:

```
struct nvldbentry *ent; // New VLDB Entry
unsigned int eix;       // Entry index
VolumeId volid;         // Volume ID
unsigned int svix; // Server index
// ... some other useful things, too
```

- `vsu_Each` calls back for each match:

```
typedef void (*vsu_each_cb)
  (struct vsu_each_answer *, void *);
```

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:
- `vsu_Each` calls back for each match:
- `struct vsu_each_query` holds query and callback:

```
struct VldbListByAttributes *vldb_attrs;
char *query; // volume name regex
afs_uint32 entry_flags_and; // VLF_*
// ...
struct cmd_item *pfx_excl;

vsu_each_cb cb; // Callback
void *cb_data; // Private pointer for cb
```

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:
- `vsu_Each` calls back for each match:
- `struct vsu_each_query` holds query and callback:
- `vsu_Each` just takes a query and runs it:
  ```
  void vsu_Each(struct vsu_each_query *q);
  ```

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:
- `vsu_Each` calls back for each match:
- `struct vsu_each_query` holds query and callback:
- `vsu_Each` just takes a query and runs it:
- Existing formatter also exported:

```
void vsu_Each_FormatAnswer
  (struct vsu_each_answer *ans, void *fmt);
```

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:
- `vsu_Each` calls back for each match:
- `struct vsu_each_query` holds query and callback:
- `vsu_Each` just takes a query and runs it:
- Existing formatter also exported:
- The `sprintf`-alike itself is available as `opr_fmt`.

Programmatic APIs

`vsu_Each` exported in `volser` API (`src/volser/volser.p.h`):

- `vos each` is a thin shim over this API.
- `struct vsu_each_answer` describes a match; VLDB entry and metadata:
- `vsu_Each` calls back for each match:
- `struct vsu_each_query` holds query and callback:
- `vsu_Each` just takes a query and runs it:
- Existing formatter also exported:
- The `sprintf`-alike itself is available as `opr_fmt`.

Take-away: every part of this effort is intended to be *reusable*.

Inside `vsu_Each`

`vsu_Each` uses `VLDB_ListAttributesN2` RPC:

- The most powerful VLDB volume query RPC to date.
- Also used by `vos listvldb` and `vos syncserv` tools.
- "Lightly documented"

VLDB_ListAttributesN2

Defined in `src/vlserver/vldbint.xg`:

```
ListAttributesN2(
    IN VldbListByAttributes *attributes,
    IN string volumename<VL_MAXNAMELEN>,
    IN  afs_int32 startindex,
    OUT afs_int32 *nentries,
    OUT nbulkentries *blkentries,
    OUT afs_int32 *nextstartindex
) = VLLISTATTRIBUTESN2;
```

In summary:

- Paged interface to bulk results.

VLDB_ListAttributesN2

Defined in `src/vlserver/vldbint.xg`:

```
ListAttributesN2(
    IN VldbListByAttributes *attributes,
    IN string volumename<VL_MAXNAMELEN>,
    IN  afs_int32 startindex,
    OUT afs_int32 *nentries,
    OUT nbulkentries *blkentries,
    OUT afs_int32 *nextstartindex
) = VLLISTATTRIBUTESN2;
```

In summary:

- Paged interface to bulk results.
- Can query by name and attributes

VLDB_ListAttributesN2 Queries

```
ListAttributesN2(
    IN VldbListByAttributes *attributes,
    IN string volumename<VL_MAXNAMELEN>,
...
```

Can query by name and attributes:

- Name is as you might expect, but
  - is a *anchored regex*!
  - must accept `.readonly` and `.backup` suffixes.
- Attribute structure contains:

```
afs_int32    server;
afs_int32    partition;
afs_uint32   volumeid;
afs_int32    flag;
```

VLDB_ListAttributesN2 Answers

Answers are expressed as "New VLDB entries":

```
char        name[VL_MAXNAMELEN];
afs_int32   nServers;
afs_int32   serverNumber[NMAXNSERVERS];
afs_int32   serverPartition[NMAXNSERVERS];
afs_int32   serverFlags[NMAXNSERVERS];
afs_uint32  volumeId[MAXTYPES];
afs_uint32  cloneId;
afs_int32   flags;
afs_int32   matchindex;
```

That is:

- Volume group name
- Each location of any volume
- The IDs of each type of volume
- $\leq$ 1 clone volume ID
- Entry flags
- Match witness

Understanding `VLDB_ListAttributesN2` *Answer*

Question: What and where is a group's RO volume?

```
char         name[VL_MAXNAMELEN];
afs_int32    nServers;
afs_int32    serverNumber[NMAXNSERVERS];
afs_int32    serverPartition[NMAXNSERVERS];
afs_int32    serverFlags[NMAXNSERVERS];
afs_uint32   volumeId[MAXTYPES];
afs_int32    flags;
```

- Check `name`.

Understanding `VLDB_ListAttributesN2` Answer

Question: What and where is a group's RO volume?

```
char        name[VL_MAXNAMELEN];
afs_int32   nServers;
afs_int32   serverNumber[NMAXNSERVERS];
afs_int32   serverPartition[NMAXNSERVERS];
afs_int32   serverFlags[NMAXNSERVERS];
afs_uint32  volumeId[MAXTYPES];
afs_int32   flags;
```

- Check `name`.
- Exists if `VLF_ROEXISTS` is set in `flags`.

Understanding `VLDB_ListAttributesN2` Answer

Question: What and where is a group's RO volume?

```
char        name[VL_MAXNAMELEN];
afs_int32   nServers;
afs_int32   serverNumber[NMAXNSERVERS];
afs_int32   serverPartition[NMAXNSERVERS];
afs_int32   serverFlags[NMAXNSERVERS];
afs_uint32  volumeId[MAXTYPES];
afs_int32   flags;
```

- Check `name`.
- Exists if `VLF_ROEXISTS` is set in `flags`.
- Volume ID is `volumeId[ROVOL]`

Understanding VLDB_ListAttributesN2 Answer

Question: What and where is a group's RO volume?

```
char        name[VL_MAXNAMELEN];
afs_int32   nServers;
afs_int32   serverNumber[NMAXNSERVERS];
afs_int32   serverPartition[NMAXNSERVERS];
afs_int32   serverFlags[NMAXNSERVERS];
afs_uint32  volumeId[MAXTYPES];
afs_int32   flags;
```

- Check `name`.
- Exists if `VLF_ROEXISTS` is set in `flags`.
- Volume ID is `volumeId[ROVOL]`
- Scan each `serverFlags` for `VLSF_ROVOL`, return corresponding `serverNumber` and `serverPartition`.

VLDB_ListAttributesN2 Sadness

There's some pain involved in this API (and the VLDB):

VLDB_ListAttributesN2 Sadness

There's some pain involved in this API (and the VLDB):

- `name` is an anchored regex:
  - using `vlserver` host regexes: `regcomp` or `re_comp`.
  - ... that still has to fit in `VL_MAXNAMELEN` bytes?!

VLDB_ListAttributesN2 Sadness

There's some pain involved in this API (and the VLDB):

- `name` is an anchored regex:
  - using `vlserver` host regexes: `regcomp` or `re_comp`.
  - ... that still has to fit in `VL_MAXNAMELEN` bytes?!
- "Clone" is not a type; it shares the `VLSF_ROVOL` flag. Cannot really tell who is hosting the clone.

VLDB_ListAttributesN2 Sadness

There's some pain involved in this API (and the VLDB):

- `name` is an anchored regex:
  - using `vlserver` host regexes: `regcomp` or `re_comp`.
  - ... that still has to fit in `VL_MAXNAMELEN` bytes?!
- "Clone" is not a type; it shares the `VLSF_ROVOL` flag. Cannot really tell who is hosting the clone.
- There's only one clone slot, while the volser has many!

VLDB_ListAttributesN2 Sadness

There's some pain involved in this API (and the VLDB):

- `name` is an anchored regex:
  - using `vlserver` host regexes: `regcomp` or `re_comp`.
  - ... that still has to fit in `VL_MAXNAMELEN` bytes?!
- "Clone" is not a type; it shares the `VLSF_ROVOL` flag. Cannot really tell who is hosting the clone.
- There's only one clone slot, while the volser has many!
- Only one IP address per server, and no UUID.

VLDB_ListAttributesN2 Sadness

- VLSF_BACKVOL not used; "know" to look for VLSF_RWVOL!
  - vos each knows and plays along (unless -noreqtyfl)
  - vsu_Each just does what it's told

VLDB_ListAttributesN2 Sadness

- VLSF_BACKVOL not used; "know" to look for VLSF_RWVOL!
  - vos each knows and plays along (unless -noreqtyfl)
  - vsu_Each just does what it's told
- Attribute entry flag field is *any set bit* triggers match.
  - Can't ask for *unset* ("unlocked") or *all of* ("RW+BK")
  - vsu_Each must post-process

VLDB_ListAttributesN2 Sadness

- VLSF_BACKVOL not used; "know" to look for VLSF_RWVOL!
  - vos each knows and plays along (unless -noreqtyfl)
  - vsu_Each just does what it's told
- Attribute entry flag field is *any set bit* triggers match.
  - Can't ask for *unset* ("unlocked") or *all of* ("RW+BK")
  - vsu_Each must post-process
- Many "entry flags" should just be "or" of server flags.
  - Opportunity for malformed replies.
  - vsu_Each by design does not check.

VLDB_ListAttributesN2 Sadness

- VLSF_BACKVOL not used; "know" to look for VLSF_RWVOL!
  - vos each knows and plays along (unless -noreqtyfl)
  - vsu_Each just does what it's told
- Attribute entry flag field is *any set bit* triggers match.
  - Can't ask for *unset* ("unlocked") or *all of* ("RW+BK")
  - vsu_Each must post-process
- Many "entry flags" should just be "or" of server flags.
  - Opportunity for malformed replies.
  - vsu_Each by design does not check.
- Match index only allows for expressing *one* match

Patch in progress

- `vos each` patchset is 1484 lines of delta:
  - 100 lines in `volser` headers
  - 626 lines for API implementation:
    - 245 for interpreting answers, 290 for formatting
    - 30 for driving `VLDB_ListAttributesN2`
  - 342 lines for `vos each` itself
    - Almost all translating `cmd` into a `vsu_each_query`.
  - 397 lines of POD docs for `vos each`.

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.
- Could now replace `vos backupsys` internals

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.
- Could now replace `vos backupsys` internals
- Could form part of a VLDB test harness
  - Stand up VLDB, synthesize changes, run queries.
  - Might allow and guide more sweeping changes to VLDB.

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.
- Could now replace `vos backupsys` internals
- Could form part of a VLDB test harness
- Looking for reviewers!
  - Much thanks to those who have already reviewed the 33 versions on gerrit.
  - Again: `http://gerrit.openafs.org/#change,10966`

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.
- Could now replace `vos backupsys` internals
- Could form part of a VLDB test harness
- Looking for reviewers!
  - Much thanks to those who have already reviewed the 33 versions on gerrit.
  - Again: `http://gerrit.openafs.org/#change,10966`

Conclusion

- Composable, friendly interfaces:
  - `vos each` CLI to `vsu_Each`
  - `vsu_Each` callback-driven API
    - And `vsu_Each_FormatAnswer` formatting callback
  - Motivated `opr_fmt` utility API.
- Could now replace `vos backupsys` internals
- Could form part of a VLDB test harness
- Looking for reviewers!
  - Much thanks to those who have already reviewed the 33 versions on gerrit.
  - Again: `http://gerrit.openafs.org/#change,10966`

Questions?