Yandex

PGBootCamp.ru

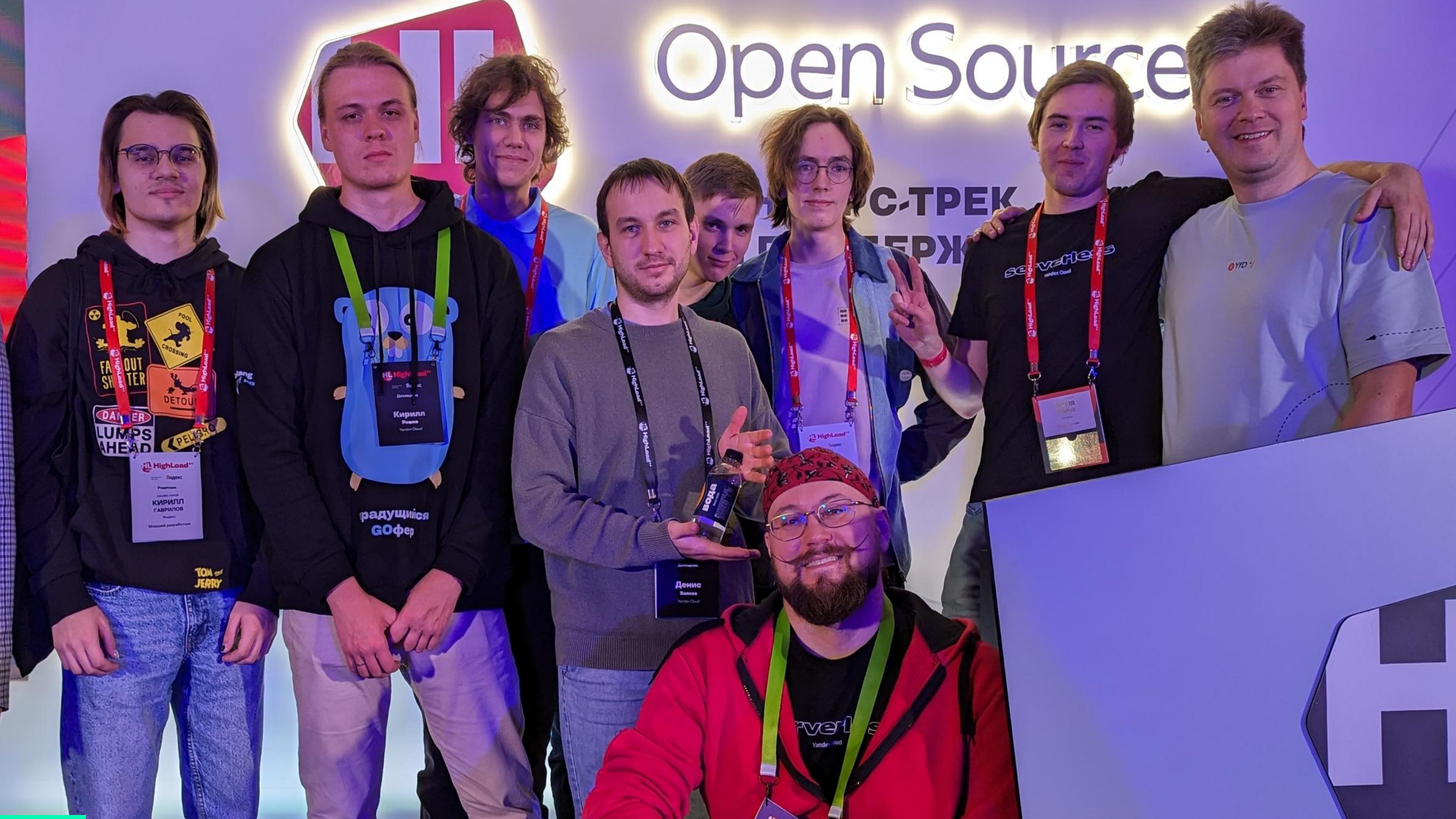
Резервное копирование с WAL-G: Анатомия catchup

Andrey Borodin, Postgres contributor

About me

- Postgres and Greenplum contributor
  on behalf of Yandex Cloud

- Maintain WAL-G, SPQR, Odyssey and some other stuff


Open Source
HighLoad++
Яндекс
Участник
КИРИЛЛ
ГАВРИЛОВ
Кирилл
Решко
Yandex Cloud
радущийся
GOфер
Денис
Волков
Yandex Cloud
ВОДА
serverless
yandex cloud
FOOL
CROSSING
DETOUR
DANGER
LUMPS
AHEAD
TOM and JERRY

Usual features

Point-in-time recovery

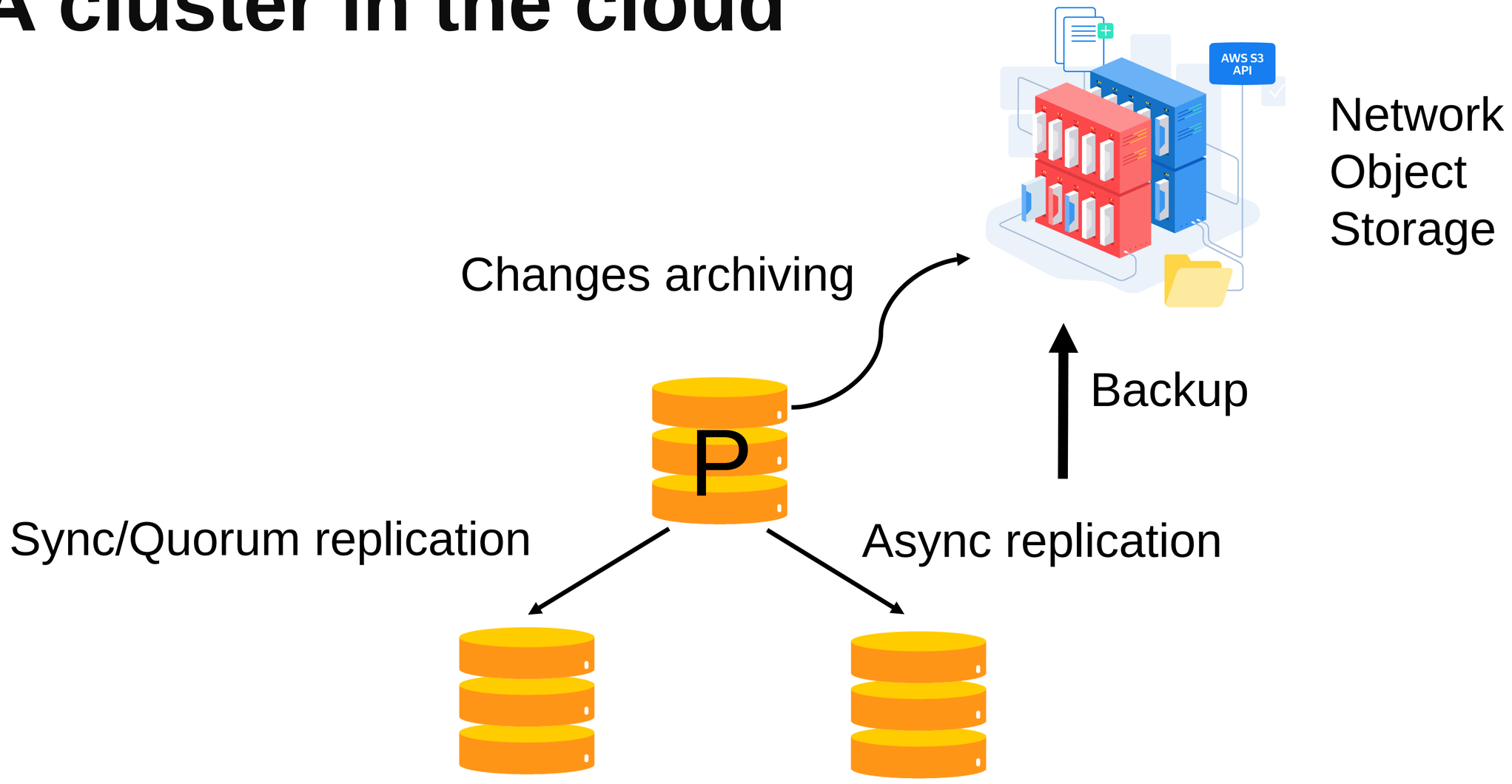
Backup + changes

- Scalable
- Reliable
- Efficient
- Fast

Scalability

- Data: from 10 GB to 10 TB on a host
- RAM: from 2 GB
- Number of CPUs: from 0.05 to ~100
- Async and parallel whenever possible
- Don’t spill anything on a local disk

HA cluster in the cloud



Resources

- Storage space

Resources

- ~~Storage space~~
- CPU
- Net bandwidth
- Disk IOPS

Reliability

- Protection from human error via automation and safety checks
- Prevention of data corruption
- Consistency monitoring
- Integration with other systems (HA tool)
- Extensibility and unification of approaches
- Encrypted data in storage

Fast recovery

- **OLAP**

From start to consistency point

- **OLTP standby**

To starting streaming replication

- **OLTP primary**

Until recovery target and accept of write queries

Unacceptable

- **Data locks**

Business can’t wait

- **Data loss**

We call it a “database” after all

pgProBackup

Barman

Backup and recovery manager for PostgreSQL

pgBackRest

WAL-G

Releases 53

v3.0.3 Latest
on Aug 8

+ 52 releases

Contributors 185

+ 171 contributors

Languages

- **Go** 88.9%
- **Shell** 9.2%
- **Other** 1.9%

3.2k stars

63 watching

457 forks

wal-g — -bash — 85×33

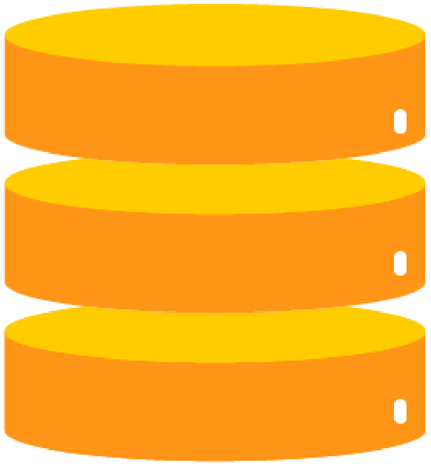
~/GoglandProjects/src/github.com/wal-g/wal-g/cmd/wal-g — -bash | ~/project/bin — psql postgres

```
x4mmm-osx:wal-g x4mmm$
x4mmm-osx:wal-g x4mmm$
x4mmm-osx:wal-g x4mmm$
x4mmm-osx:wal-g x4mmm$ AWS_ENDPOINT=https://storage.yandexcloud.net AWS_ACCESS_KEY_ID
=wIRAxxwOPLI3VrGwtYWL AWS_SECRET_ACCESS_KEY=ne[redacted]vsXX
WALE_S3_PREFIX=s3://wal-g-test/ ./wal-g backup-list
Path:
name                          last_modified        wal_segment_backup_start
base_000000010000000000000004 2019-02-02T18:39:30Z 000000010000000000000004
x4mmm-osx:wal-g x4mmm$
x4mmm-osx:wal-g x4mmm$
```

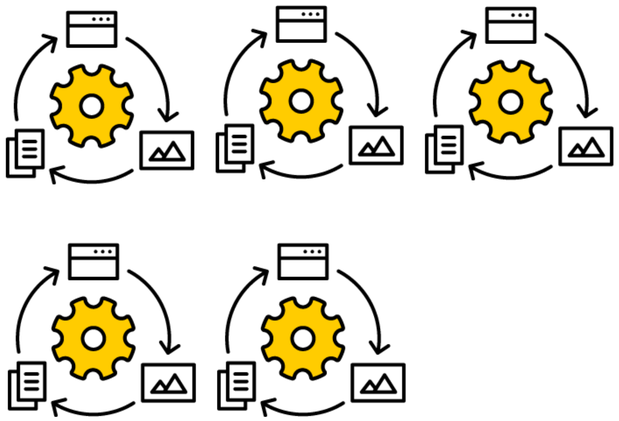
```
x4mmm-osx:wal-g x4mmm$
x4mmm-osx:wal-g x4mmm$ AWS_ENDPOINT=https://storage.yandexcloud.net AWS_ACCESS_KEY_ID
=wIRAxxwOPLI3VrGwtYWL AWS_SECRET_ACCESS_KEY=neh7EEYANpqGS5GJEbmOywhznxcIBukG3IamvsXX
WALE_S3_PREFIX=s3://wal-g-test/ ./wal-g backup-push ~/DemoDb
Path:
INFO: 2019/02/02 21:56:42.509465 Doing full backup.
WARNING: 2019/02/02 21:56:42.526434 It seems your archive_mode is not enabled. This w
ill cause inconsistent backup. Please consider configuring WAL archiving.
INFO: 2019/02/02 21:56:42.740377 Walking ...
INFO: 2019/02/02 21:56:42.742571 Starting part 1 ...
INFO: 2019/02/02 21:56:43.112485 Finished writing part 1.
INFO: 2019/02/02 21:56:48.744337 Starting part 2 ...
INFO: 2019/02/02 21:56:48.761395 /global/pg_control
INFO: 2019/02/02 21:56:48.764006 Finished writing part 2.
INFO: 2019/02/02 21:56:48.878931 Starting part 3 ...
INFO: 2019/02/02 21:56:48.894990 backup_label
INFO: 2019/02/02 21:56:48.895030 tablespace_map
INFO: 2019/02/02 21:56:48.895056 Finished writing part 3.
INFO: 2019/02/02 21:56:49.523658 Uploaded 3 compressed tar Files.
x4mmm-osx:wal-g x4mmm$
```

```
# - Archiving -

archive_mode = on

archive_command = '/usr/bin/envdir /etc/wal-g/envdir
/usr/bin/timeout 600 /usr/bin/wal-g wal-push %p'
```

Base backup



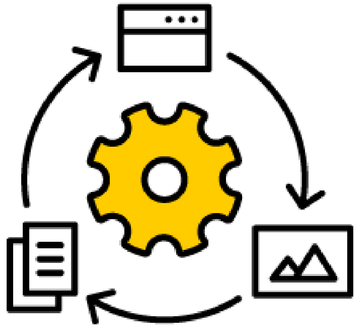
DB copy



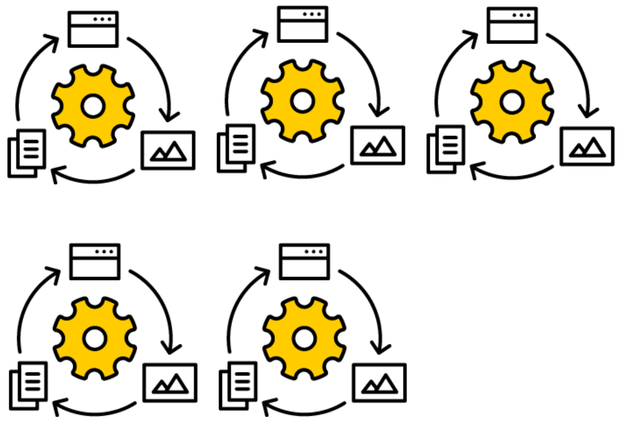
Changes (WAL)

Delta backups

DB copy

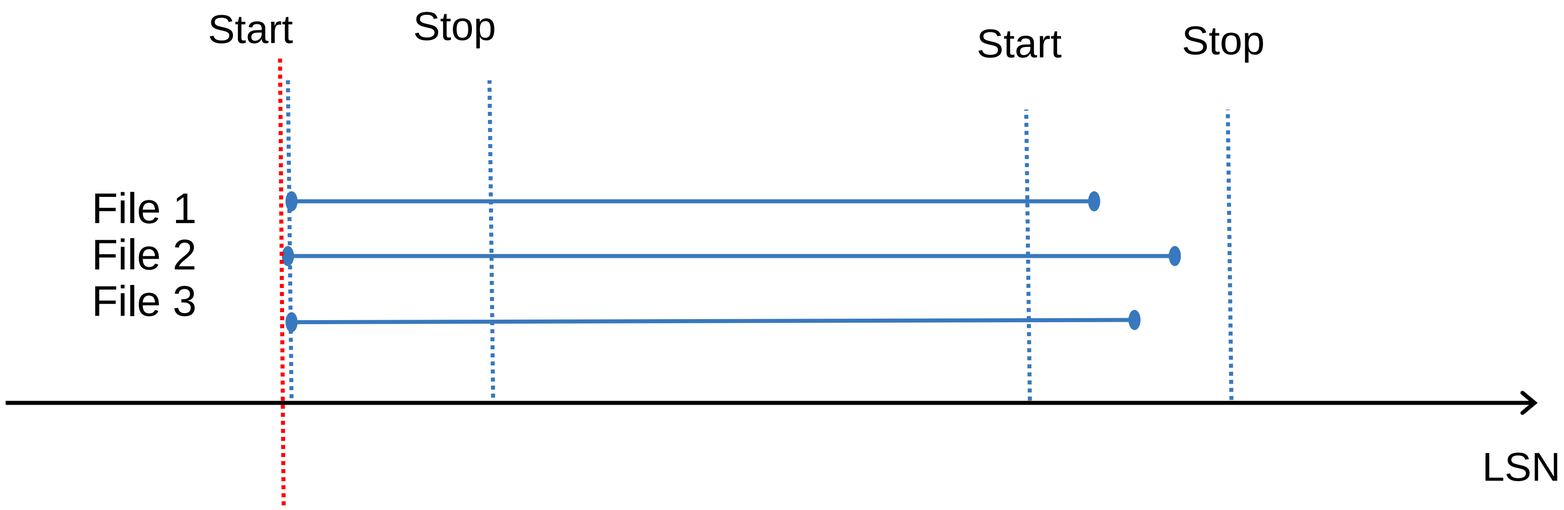


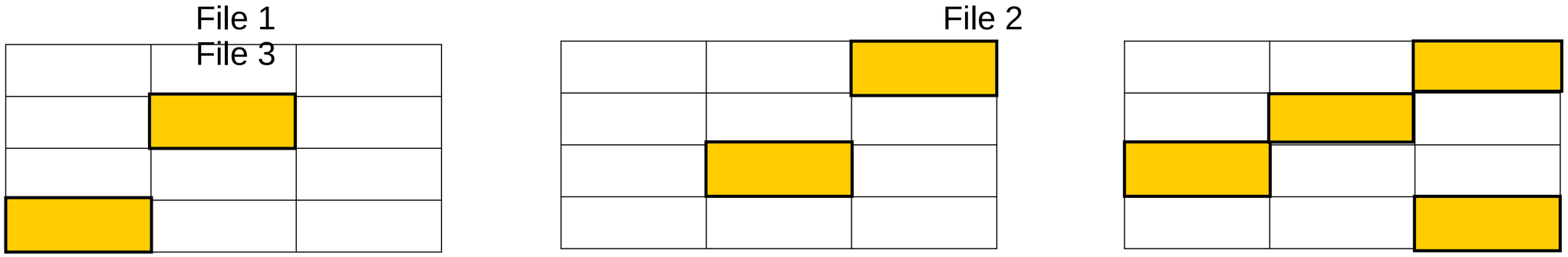
Delta copy



Changes (WAL)

LSN-based deltas





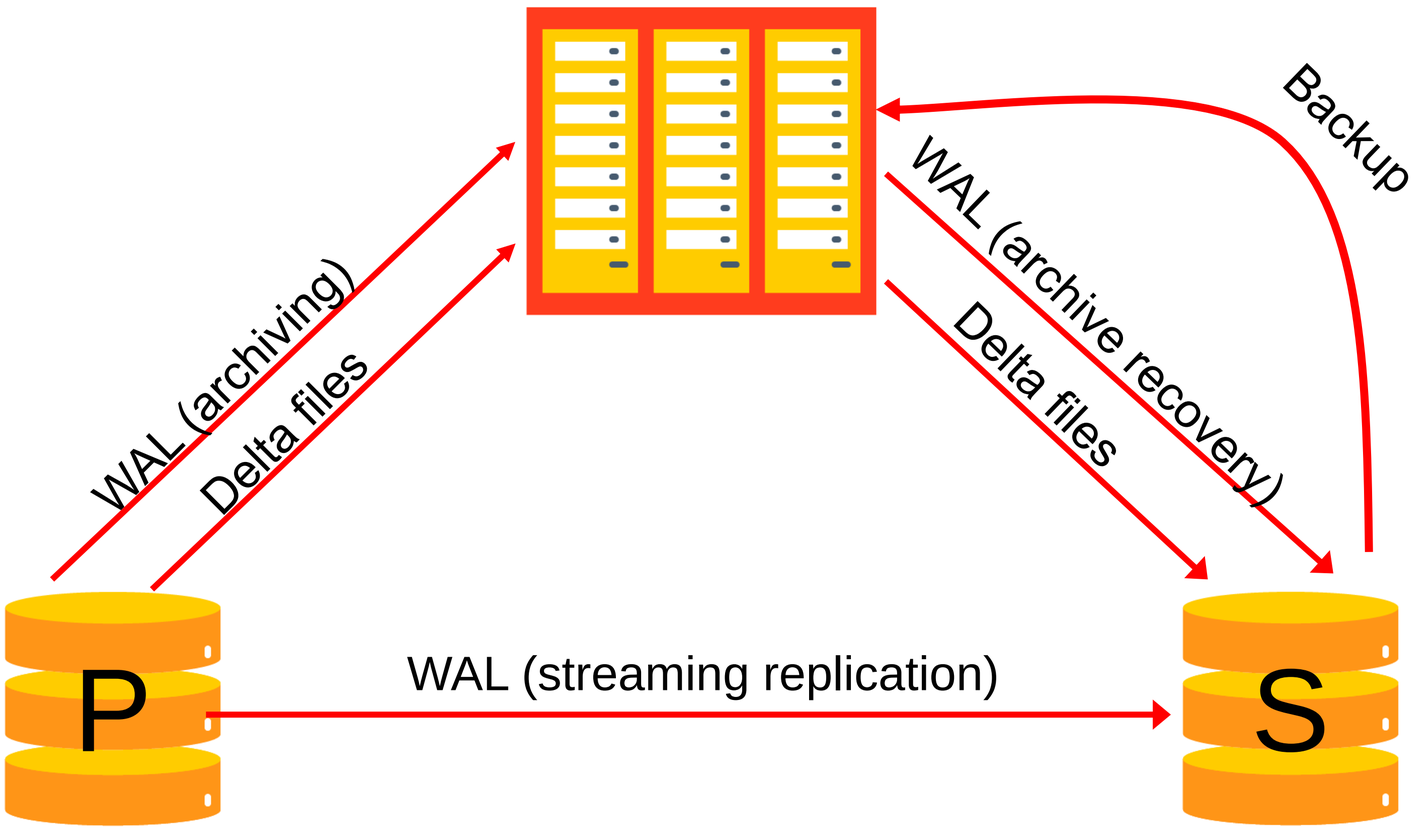
Unusual features

Parallel WAL interface

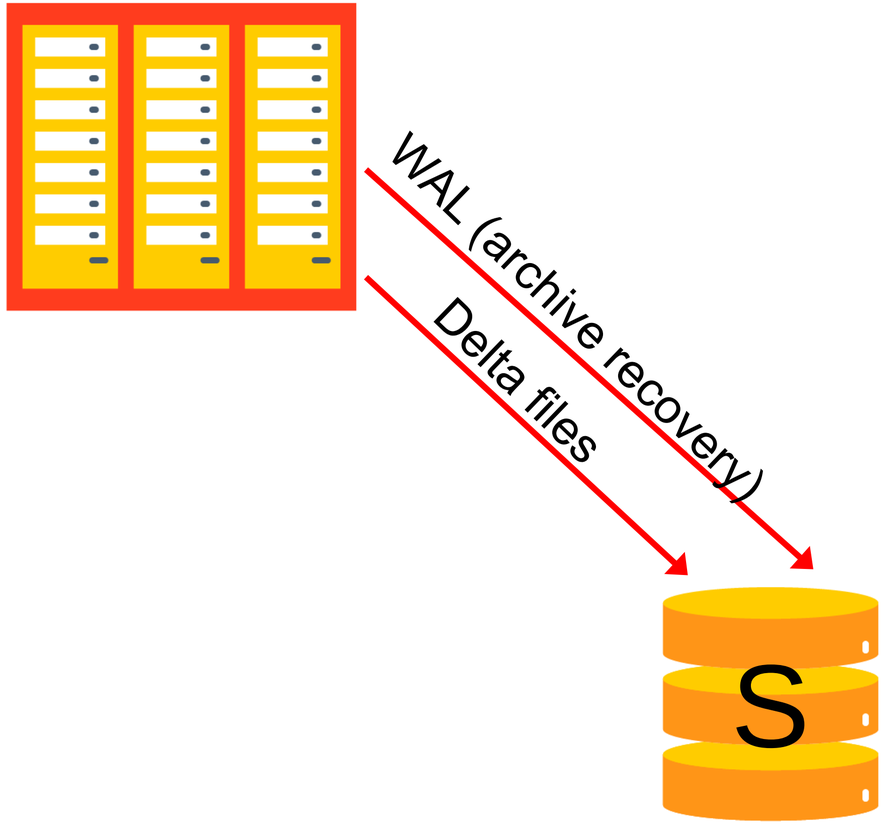
- archive_command
- restore_command

Synchronous PG calls are triggering prefetches

Data flows in the system



WAL-based prefault



Throttling

WALG_NETWORK_RATE_LIMIT
WALG_DISK_RATE_LIMIT

Throttling

WALG_NETWORK_RATE_LIMIT
WALG_DISK_RATE_LIMIT

- WALG_UPLOAD_DISK_CONCURRENCY

Throttling

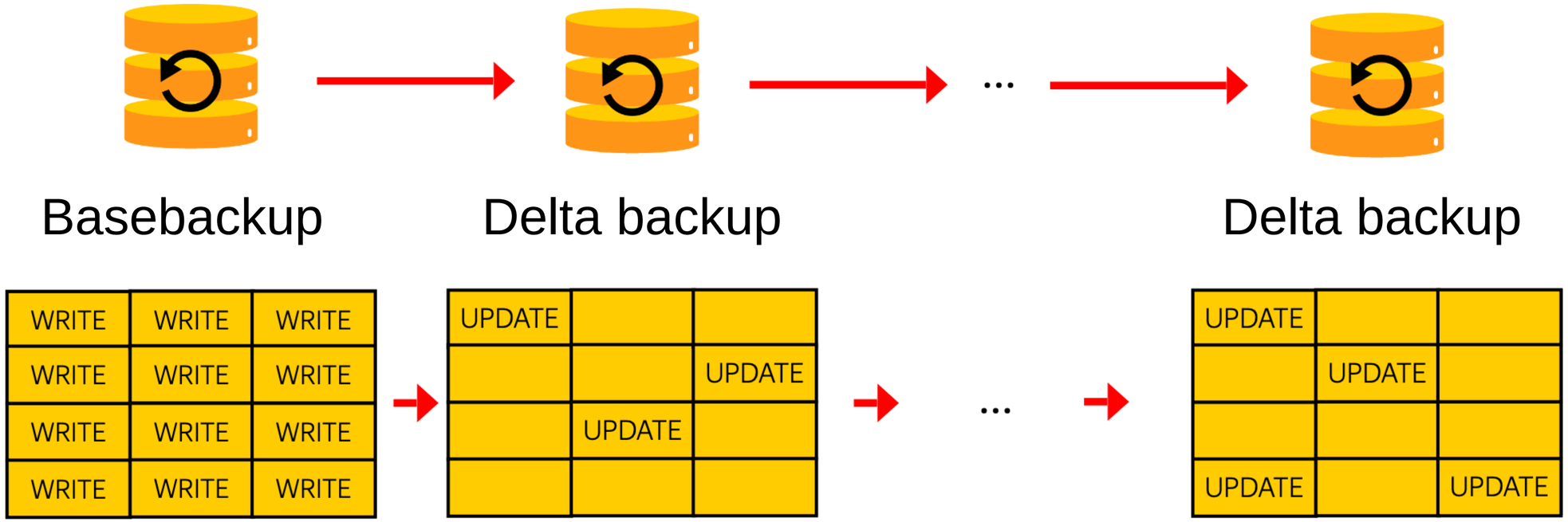
WALG_NETWORK_RATE_LIMIT
WALG_DISK_RATE_LIMIT

- WALG_UPLOAD_DISK_CONCURRENCY

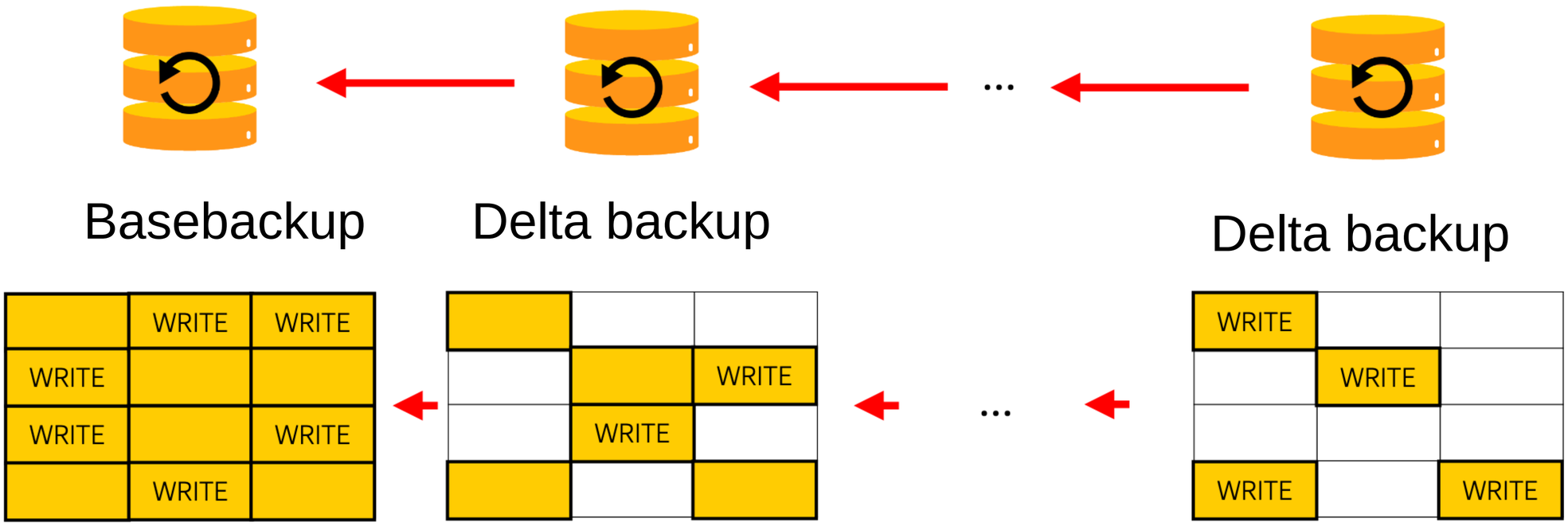
```
wal-g backup-push --turbo --config=/usual/wal-g.yaml
```

Delta unpack

Direct delta unpack



Reverse delta unpack




Disk I/O (system.io)
REVERSE
STANDARD
proc:/proc/diskstats
system.io
MiB/s
in
out
MiB/s
0,0
-19,5
-39,1
-58,6
-78,1
00:15:00
00:20:00
00:25:00
00:30:00
00:35:00
00:40:00
00:45:00
00:50:00

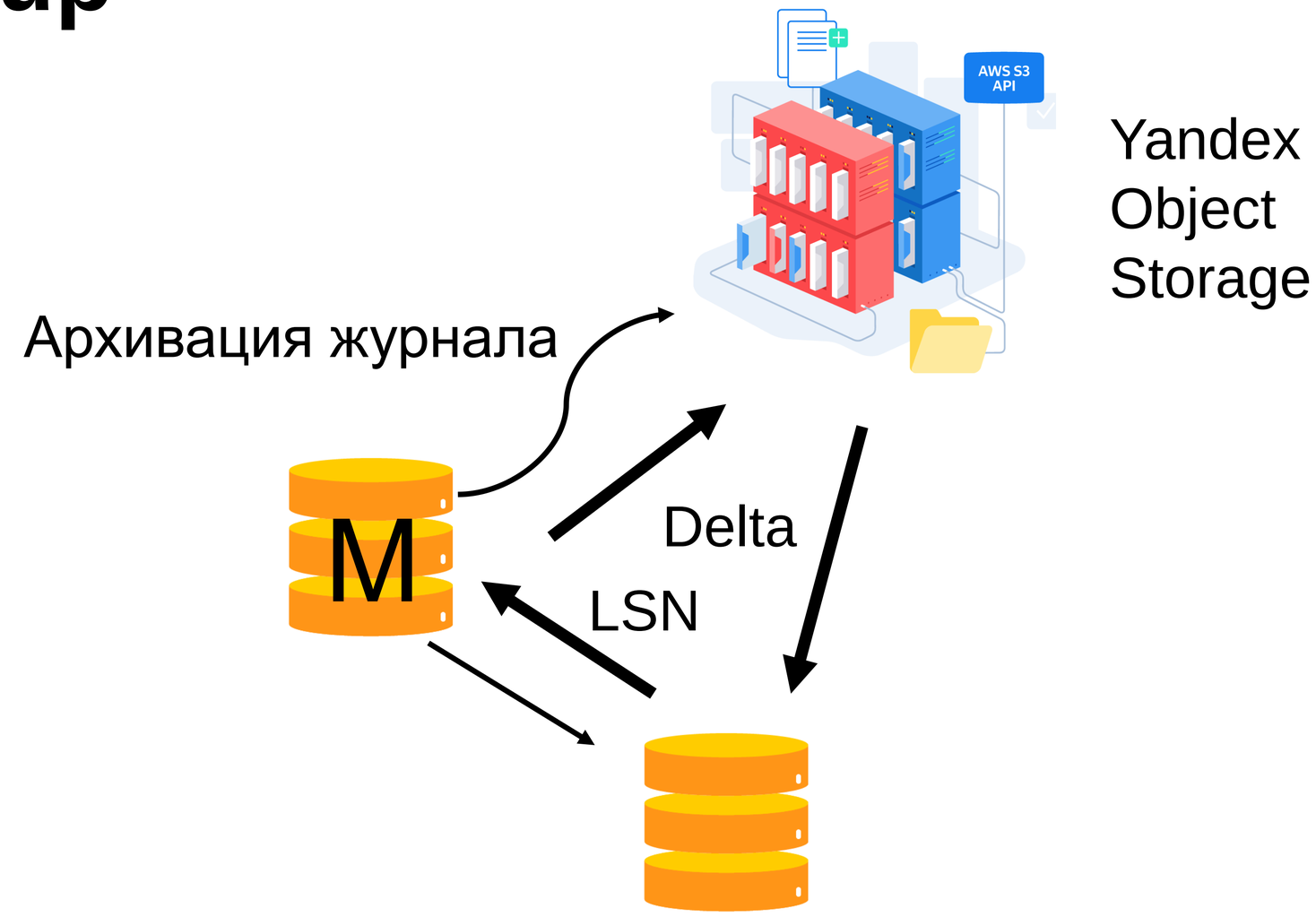
Catchup

Oracle: Rolling forward a standby database using RMAN

In this Document

Goal
Solution
1) On the standby database, stop the managed recovery process (MRP)
2) On the standby database, find the SCN which will be used for the incremental backup at the primary database:
3) In sqlplus, connect to the primary database and identify datafiles added:
4) Using rman, create backup of missing datafiles and an incremental backup using the SCN derived in the previous step:
5) Transfer all backup sets created on the primary system to the standby system.
6) Restore new controlfile and catalog the backup transfered in step #5:
7) Restore missing datafiles:
8) Rename the datafiles in new standby controlfile
9) Recover the standby database with the cataloged incremental backup:
10) If the standby database needs to be configured for FLASHBACK use the below step to enable.
11) On standby database, clear all standby redo log groups:
12) On the standby database, start the MRP
References

Catchup


AWS S3
API
Yandex
Object
Storage
Архивация журнала
M
Delta
LSN

WAL-G

```
wal-g catchup-send        # on Primary
wal-g catchup-receive     # on Standby
```

- Beware: wal-g must open tcp port on primary, not 5432\6432

Features

- Encrypted with WAL-G settings
- Safety checks
- Idempotent

Features

- Encrypted with WAL-G settings
- Safety checks
- Idempotent (TODO: move pg_control)

Features

- Encrypted with WAL-G settings
- Safety checks
- Idempotent (TODO: move pg_control)

Parallelism is not implemented yet

Under the Hood

- catchup_fetch_handler.go
- catchup_file_unwrapper.go
- catchup_push_handler.go
- catchup_send_recieve_handler.go

catchup_send_recieve_handler.go 1

internal > databases > postgres > catchu

```go
func receiveFileList(directory
	})
	tracelog.ErrorLogger.Fatal(
	return result
}
```

Receiver

```go
func HandleCatchupReceive(pgDataDirectory string, port int) {
	listen, err := net.Listen("tcp", fmt.Sprintf(":%v", port))
	conn, err := listen.Accept()

	sendControlAndFileList(pgDataDirectory, encoder)

	for {
		var cmd CatchupCommandDto
		err := decoder.Decode(&cmd)
		tracelog.ErrorLogger.FatalOnError(err)
		if cmd.IsDone {
			break
		}
		doRcvCommand(cmd, pgDataDirectory, decoder)
	}
	tracelog.InfoLogger.Printf("Receive done")
}
```

Receiver: TODO

```go
func HandleCatchupReceive(pgDataDirectory string, port int) {
	listen, err := net.Listen("tcp", fmt.Sprintf(":%v", port))
	conn, err := listen.Accept()

	sendControlAndFileList(pgDataDirectory, encoder)

	for {
		var cmd CatchupCommandDto
		err := decoder.Decode(&cmd)
		tracelog.ErrorLogger.FatalOnError(err)
		if cmd.IsDone {
			break
		}
		doRcvCommand(cmd, pgDataDirectory, decoder)
	}
	tracelog.InfoLogger.Printf("Receive done")
}
```

Must be parallel!

Receiver: TODO

```go
func HandleCatchupReceive(pgDataDirectory string, port int) {
    listen, err := net.Listen("tcp", fmt.Sprintf(":%v", port))
    conn, err := listen.Accept()

    sendControlAndFileList(pgDataDirectory, encoder)

    for {
        var cmd CatchupCommandDto
        err := decoder.Decode(&cmd)
        tracelog.ErrorLogger.FatalOnError(err)
        if cmd.IsDone {
            break
        }
        doRcvCommand(cmd, pgDataDirectory, decoder)
    }
    tracelog.InfoLogger.Printf("Receive done")
}
```

What if we have a crash?

Sender

```go
22  func HandleCatchupSend(pgDataDirectory string, destination string) {
23      info, runner, err := GetPgServerInfo(true)
24      writer, decoder, encoder := startSendConnection(destination)
25
26      var control PgControlData
27      err = decoder.Decode(&control)
28      tracelog.InfoLogger.Printf("Destination control file %v", control)
29      tracelog.InfoLogger.Printf("Our system id %v", *info.systemIdentifier)
30      if *info.systemIdentifier != control.SystemIdentifier {
31          tracelog.ErrorLogger.Fatal("System identifiers do not match")
32      }
33      if control.CurrentTimeline != info.Timeline {
34          tracelog.ErrorLogger.Fatalf("Destination is on timeline %v, but we are on %v",
35              control.CurrentTimeline, info.Timeline)
36      }
37
38      var fileList internal.BackupFileList
39      err = decoder.Decode(&fileList)
40      if lsn <= control.Checkpoint {
41          tracelog.ErrorLogger.Fatalf("Catchup destination is already ahead (our LSN %v, destination LSN %v).",
42              lsn, control.Checkpoint)
43      }
```

Sender

```go
	sendFileCommands(encoder, pgDataDirectory, fileList, control.Checkpoint)


	label, offsetMap, _, err := runner.StopBackup()

	err = encoder.Encode(
		CatchupCommandDto{BinaryContents: []byte(label), FileName: BackupLabelFilename, IsBinContents: true})
	err = encoder.Encode(
		CatchupCommandDto{BinaryContents: []byte(offsetMap), FileName: TablespaceMapFilename, IsBinContents: true})

	ourPgControl, err := os.ReadFile(path.Join(pgDataDirectory, PgControlPath))
	err = encoder.Encode(
		CatchupCommandDto{
			BinaryContents: ourPgControl, FileName: utility.SanitizePath(PgControlPath), IsBinContents: true,
		})

	err = encoder.Encode(CatchupCommandDto{IsDone: true})

	tracelog.InfoLogger.Printf("Send done")
}
```

TODO

- Use WAL Delta map
- More checks
- More tests!!!

Tests!

```bash
#!/bin/bash
set -e -x

# init alpha cluster
/usr/lib/postgresql/10/bin/initdb ${PGDATA_ALPHA}
/usr/lib/postgresql/10/bin/pg_ctl -D ${PGDATA_ALPHA} -w start

# init beta cluster (replica of alpha)
/usr/lib/postgresql/10/bin/pg_basebackup --wal-method=stream -D ${PGDATA_BETA} -U repl -h 127.0.0.1 -p ${ALPHA_PORT}
/usr/lib/postgresql/10/bin/pg_ctl -D ${PGDATA_BETA} -w start

# fill database postgres
pgbench -i -s 15 -h 127.0.0.1 -p ${ALPHA_PORT} postgres
/usr/lib/postgresql/10/bin/pg_ctl -D ${PGDATA_BETA} --mode smart -w stop
pgbench -T 10 -P 1 -h 127.0.0.1 -p ${ALPHA_PORT} postgres

# create some new files
pgbench -i -s 5 -h 127.0.0.1 -p ${ALPHA_PORT} postgres
/usr/lib/postgresql/10/bin/pg_dump -h 127.0.0.1 -p ${ALPHA_PORT} -f ${ALPHA_DUMP} postgres
```

Tests!

```
21  wal-g catchup-receive ${PGDATA_BETA} 1337 &
22
23  wal-g --config=${TMP_CONFIG} catchup-send ${PGDATA_ALPHA} localhost:1337
24
25  /usr/lib/postgresql/10/bin/pg_ctl -D ${PGDATA_BETA} -w start
26
27  /usr/lib/postgresql/10/bin/pg_dump -h 127.0.0.1 -p ${BETA_PORT} -f ${BETA_DUMP} postgres
28
29  diff ${ALPHA_DUMP} ${BETA_DUMP}
```

TODO

- Check edge cases

Replica ahead master
Replica of another cluster

- Check failure during catchup

- Stress-test

PGBootCamp.ru

Thanks! ☺

Andrey Borodin

Postgres contributor

x4mmm @yandex-team.ru

x4mmm