Moving PostgreSQL Forward – pgconf.in

Robert Haas

PostgreSQL Features

- Usually High Quality
- Often Delivered Incrementally Across Multiple Releases
- Sometimes Slow To Arrive

Why?

What Happens When a Feature is Buggy?

- Frustrating user experience.
- Poor performance.
- Wrong answers.
- Server crashes.
- Data loss.

- Development resources redirected to stabilization.
- Damage to project reputation.

PostgreSQL Philosophy

- Getting it done right is more important than getting it done sooner.

Can we have both?

Important Caveat: Accelerating Progress

- Logical Decoding (9.4)
- JSONB (9.4)
- INSERT .. ON CONFLICT (9.5)
- Parallel Query (9.6)
- Partitioning (10)
- Logical Replication (10)
- Improved Parallel Query (10)
- Performance Improvements (every release)
- Test Framework Improvements (every release)

But Still … Faster Is Better!

- Other database systems are continuing to innovate at a rapid pace.
- Some features which PostgreSQL has recently added, such as declarative partitioning, logical replication, and parallel query have been present in some commercial database systems for decades.
- Our goal should be to innovate as quickly as possible, provided that we can do it without compromising quality.

Why Can't We Make Progress Faster?

- Insufficient Patch Quality (as judged by author, reviewers, committer)
- Political Opposition (especially, but not only, from committers)
- Patches Typically Have One Or Two Primary Authors
- Development Community Isn't All That Large
- Shortage of Committer Bandwidth
- Patch Author Gives Up

Statistics Methodology

- Use `git log` to identify commits from 2016
- Used `-w` option to suppress whitespace-only changes
- Used `-M` option to suppress changes due to file renames
- Eliminated large mechanical commits, primarily translation updates
- Recorded lines of new code, as per `--stat`
- Manually identified primary patch author
- Imported results into PostgreSQL database

Development Community Statistics (2016)

- 141 people contributed at least 1 line of new code.
- 90% of new lines of code were written by 37 people.
- 66% of new lines of code were written by 14 people.

- 18 committers committed at least 1 patch by a non-committer.
- 90% of new lines of non-committer code were committed by 6 committers.
- 66% of new lines of non-committer code were committed by 2 committers.

Top Primary Patch Authors (2016)

#	author	total_lines	percentage_lines	total_commits
1	Tom Lane	62077	29.20	637
2	Amit Langote [*]	9889	4.65	30
3	Robert Haas	9685	4.55	108
4	Stephen Frost	9177	4.32	46
5	Teodor Sigaev	8345	3.92	28
6	Michael Paquier [*]	7778	3.66	106
7	Andres Freund	5913	2.78	61
8	David Rowley [*]	5582	2.63	26
9	Alexander Korotkov [*]	5174	2.43	11
10	Peter Eisentraut	4877	2.29	161
11	Heikki Linnakangas	4378	2.06	42
12	Thomas Munro [*]	3535	1.66	31
13	Magnus Hagander	3494	1.64	26
14	Amit Kapila [*]	3480	1.64	35
15	Kevin Grittner	3103	1.46	23
16	Andreas Karlsson [*]	3062	1.44	33
17	Bruce Momjian	3049	1.43	27
18	Fabien Coelho [*]	2768	1.30	22
19	Shigeru Hanada [*]	2752	1.29	3
20	Alvaro Herrera	2636	1.24	56
21	Jeevan Chalke [*]	2454	1.15	2
22	Etsuro Fujita [*]	2378	1.12	21
23	Kyotaro Horiguchi [*]	2171	1.02	19
24	Masahiko Sawada [*]	2129	1.00	20
25	Peter Geoghegan [*]	2121	1.00	25

Committers of Others' Patches (2016)

#	committer	total_lines	percentage_lines	total_commits
1	Robert Haas	37726	40.03	241
2	Tom Lane	25293	26.84	204
3	Alvaro Herrera	7611	8.08	59
4	Teodor Sigaev	7252	7.70	32
5	Heikki Linnakangas	4191	4.45	33
6	Peter Eisentraut	3588	3.81	56
7	Andres Freund	2558	2.71	22
8	Simon Riggs	1886	2.00	21
9	Fujii Masao	1626	1.73	21
10	Magnus Hagander	638	0.68	30
11	Noah Misch	533	0.57	10
12	Andrew Dunstan	426	0.45	6
13	Kevin Grittner	401	0.43	8
14	Stephen Frost	381	0.40	6
15	Dean Rasheed	56	0.06	1
16	Bruce Momjian	49	0.05	10
17	Joe Conway	23	0.02	1
18	Michael Meskes	5	0.01	1

What Do These Statistics Tell Us?

- The PostgreSQL development community is fairly small.
- Even in theory though anyone can submit a patch, in practice nearly all work is done by a few dozen people.
- Tom Lane's output is exceptional, personally writing more than six times as much PostgreSQL as anyone else, and also committing more non-committer code than anyone except me.
- If you can't catch the interest of one of the small number of active committers, getting your patch committed is hard.

Some Patches Pending For a While...

- SCRAM Authentication (originally submitted to CommitFest 2015-09)
- Fix the optimization to skip WAL-logging on table created in the same transaction (originally submitted to CommitFest 2016-03)
- Unique Joins (originally submitted to CommitFest 2015-02)
- Multivariate Statistics (originally submitted to CommitFest 2016-01)
- amcheck (originally submitted to CommitFest 2016-03)

Do We Just Need More Committers?

- Actually committing a patch is easy; what's hard is reviewing a patch.
- If we had a supply of expert-level reviewers with adequate time to spend on patch review, we could make them committers!
- Becoming an expert is hard, especially if you can't get feedback on your own patches.
- This is a chicken and egg problem.

Why Can't We Make Progress Faster?

- Insufficient Patch Quality (as judged by author, reviewers, committer)
- Political Opposition (especially, but not only, from committers)
- Patches Typically Have One Or Two Primary Authors
- Development Community Isn't All That Large
- Shortage of Committer Bandwidth
- Patch Author Gives Up

Why Can't We Make Progress Faster?

- Insufficient Patch Quality (as judged by author, reviewers, committer)
- Political Opposition (especially, but not only, from committers)
- Patches Typically Have One Or Two Primary Authors
- Development Community Isn't All That Large
- Shortage of Committer Bandwidth
- Patch Author Gives Up

Small Development Community

- Insufficient Patch Quality (as judged by author, reviewers, committer)
  - There are few people who are experts to write the patches, and anybody who is not an expert will find it difficult.
- Political Opposition (especially, but not only, from committers)
  - One or two key people can block a patch.
- Patches Typically Have One Or Two Primary Authors
  - It's hard to assemble a big team from a small community.
- Development Community Isn't All That Large
- Shortage of Committer Bandwidth
  - It's hard to find qualified committers in a small pool of developers.
- Patch Author Gives Up
  - The existing developers are very busy, so new people don't get much coaching.

How Do We Fix It?

- We need to involve more people in PostgreSQL development! Eventually, some of them will become experts who can expand the pool of committers.
- Experienced contributors need to write patch reviews with the goal of helping less-experienced contributors learn – and also encourage them to persist!
- Companies that care about PostgreSQL should pay their employees to help with patch review and commit.
- Working in teams can make it possible to tackle larger projects.
- You tell me?

Case Study – Parallel Query

- Core of parallel query was mostly designed by Robert Haas and Noah Misch and mostly coded by Robert Haas and Amit Kapila, with some help from others.
- Once we had the basics (Gather, Parallel Seq Scan), it became possible for more people to contribute.
- In 9.6, David Rowley contributed parallel aggregate, Constantin Pan submitted a patch for parallel GIN index build, and Andreas Karlsson did parallel-safety.
- In 10, several people have submitted patches for parallel utility commands – Peter Geoghegan has worked on parallel btree index creation, Masahiko Sawada on parallel vacuum.

Case Study – Parallel Query

- Also in 10, EnterpriseDB employees submitted patches for:
  - Parallel Bitmap Heap Scan (Dilip Kumar)
  - Parallel Index Scan (Rahila Syed, Amit Kapila)
  - Parallel Index-Only Scan (Rafia Sabih)
  - Parallel Shared Hash (Thomas Munro)
  - Parallel Append (Amit Khandekar)
  - Gather Merge (Rushabh Lathia)
  - Enabling parallelism for Merge Joins (Dilip Kumar)
  - Enabling parallelism for SubPlans (Amit Kapila)
  - Enabling parallelism for InitPlans (Amit, Kuntal)
  - Enabling parallelism at serializable (Thomas Munro)
  - Enabling parallelism in PLs (Rafia Sabih)

Case Study – Hash Index

- EnterpriseDB made a strategic decision to work towards making hash indexes write-ahead logged.
- By necessity, most of the work had to be done by a single individual, because it was essential to have a single vision driving the work. So Amit Kapila did most of the work.
- However, he was assisted by Kuntal Ghosh (WAL consistency checker, stress testing), Mithun Cy (metapage cache, better expansion), and Ashutosh Sharma (pgstattuple and pageindex support, page scan mode, microvacuum support).

Lessons Learned

- EnterpriseDB was able to speed up projects by finding ways for multiple developers to work in the same general area without interfering with each others' work.
- Building tools for benchmarking and stress-testing improves outcomes and makes patches more reliable, speeding up progress.
- However, finding reviewers remains a challenge. It will take time to grow the development community.
- Writing your first substantial patch is always hard! You'll probably make lots of mistakes...

Hopes For the Future

- More developers.
- More reviewers.
- More committers.
- More multi-person projects.
- More cross-company initiatives.
- PostgreSQL world domination!

Thank You

- Any questions?