Debian Code Search Instant

GPN15, 2015-06-05

Michael Stapelberg
<stapelberg@debian.org>

Kurze Wiederholung

- Vortrag auf der GPN13 (Video und Folien)
- Motivation (warum Code-Suchmaschine?)
- Regexp-Suche basierend auf rsc's `codesearch`
- Hosting auf meinem Privat-Server
- Index 1x/Woche manuell aktualisiert


Debian Code Search

Agenda

- Nutzen in FOSS/Debian
- Monitoring
- Alte/Neue Hardware (Rackspace)
- Alte/Neue Architektur / Instant Indexing
- Streaming Queries / Instant Results
- Optimierung
- Ausblick

Nutzen in FOSS/Debian

- avg. 125 Suchanfragen täglich (seit 2014-12)

- 12 Testimonials auf wiki:DebianCodeSearch
  (von Menschen aus Debian, Tor, OpenBSD, …)
  „codesearch is becomming an essential part of the Debian infrastructure, that allows one to quickly answer questions that would otherwise have needed tedious greps in the Lintian lab. […]“ — Charles Plessy, Debian Developer

Motivation für Verbesserungen

- Top feature request:
  Suchergebnisse nach Paket gruppieren

- Top bug:
  60s-Timeout für Suchanfragen
  (macht manche Anfragen unmöglich)

Monitoring

- Prometheus (Vortrag Sa, 2015-06-06 17:30)

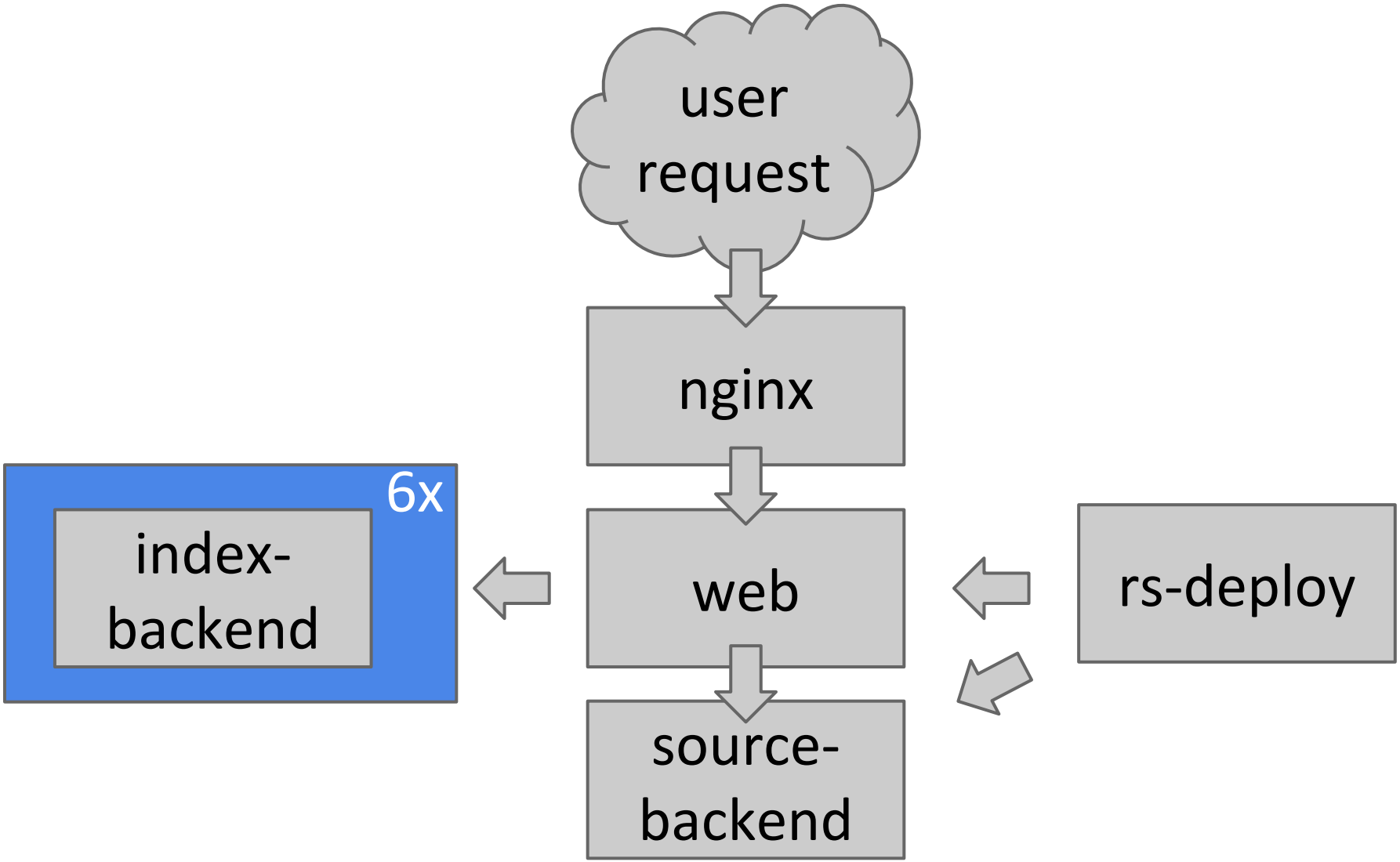
- Prozesse exportieren Metriken
  #source-packages, #failures, …

- <demo>

Hardware (Rackspace)

- Rackspace sponsert DCS seit Ende 2013
- Trigram Index auf 6 Rechner verteilt,
  Source auf Block Storage (an 1 Rechner)
- Jede Woche komplett neu deployed
- Nicht mit debmirror: 1 GBit/s Link auslasten

Alte Architektur



Neue Hardware (Rackspace)

- 4 × Intel® Xeon®E5-2670
  4 GiB RAM
  80 GiB SSD (> 5000 IOPS)
  800 MBit/s Netzanbindung

- 6 Instanzen für Index/Source

- 1 Instanz für Frontend, 1 Instanz für Monitoring

Neue Architektur (1)

- „Instant Indexing“-Idee:
  Index nicht wöchentlich komplett neu bauen,
  sondern Paket-weise (häufig) aktualisieren

- Problem: Indexformat nicht für inkrementelle
  Updates ausgelegt

Neue Architektur (2)

- 100% overhead akzeptabel durch neue HW
  → $package.idx und full.idx behalten

- effizienter 5-Minuten-Merge implementiert

- dcs-package-importer liest ein Debian-Paket, entpackt, indexiert, merged nach full.idx

- dcs-index-backend kann neuen full.idx laden

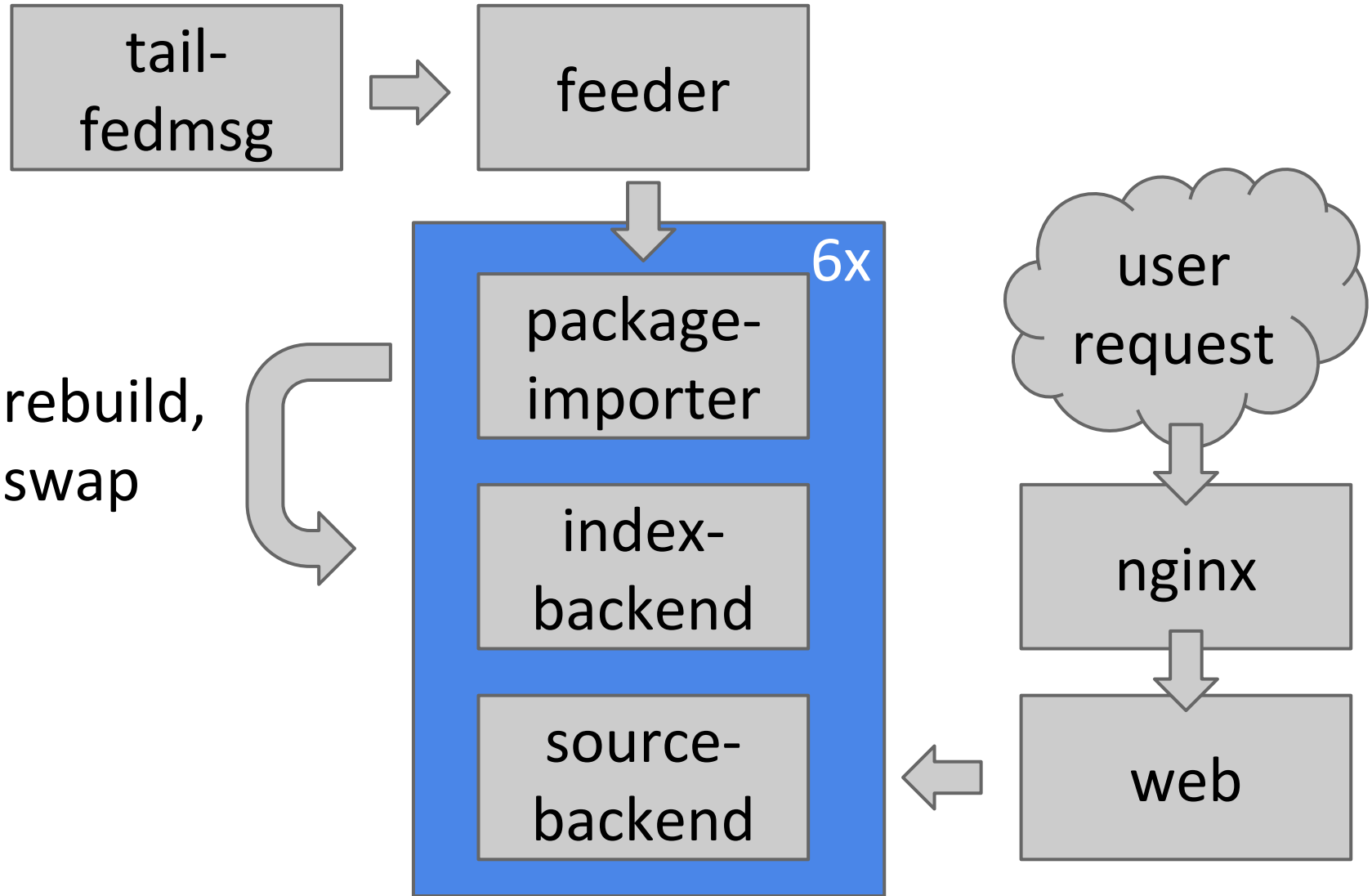
Neue Architektur (3)

- dcs-feeder pollt stündlich neue Packages,
  lädt Package in passenden dcs-package-importer

- dcs-tail-fedmsg benachrichtigt dcs-feeder bei neuen Uploads (via FedMsg)

- dcs-source-backend fragt lokales dcs-index-backend an und führt Suche durch

Neue Architektur (3)



Streaming Queries (1)

- Früher: intern je 1000 Ergebnisse geranked und seitenweise angezeigt
- Heute: Ranking erst bestimmt wenn Suche fertig, aber: 10 beste Ergebnisse werden aktualisiert
- Kein Warten mehr auf alle Index-Treffer

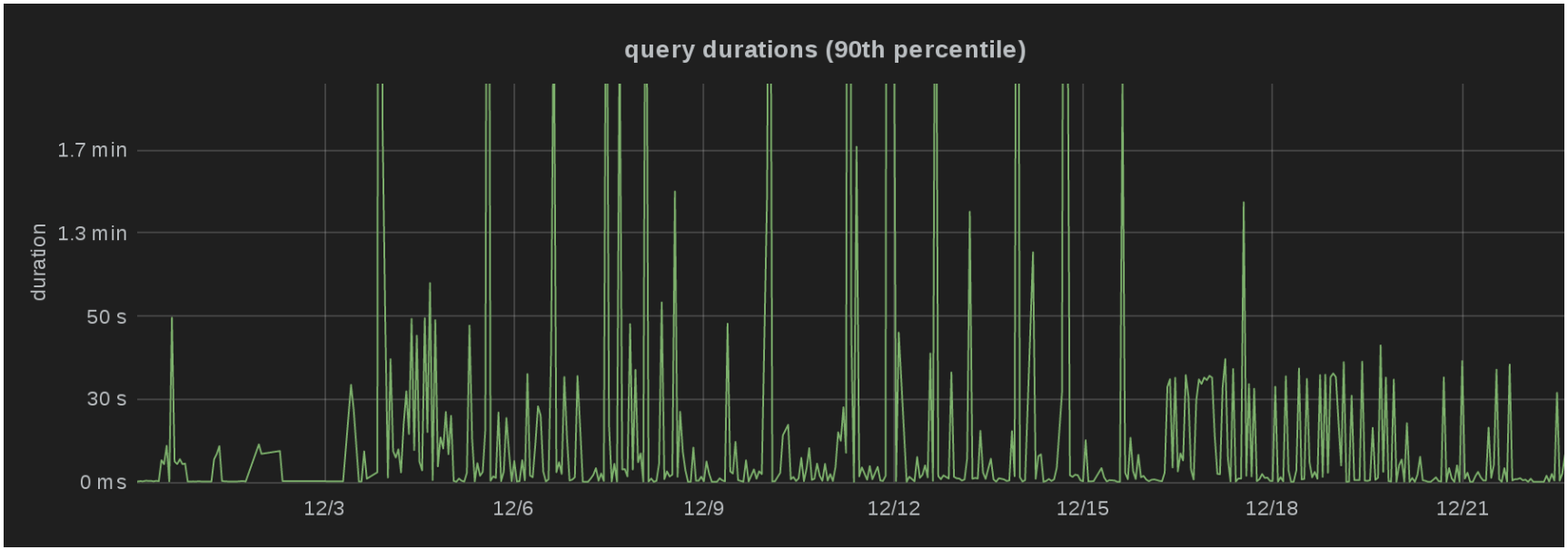
Streaming Queries (2)

- Websockets
- Meta-Info (Fortschritt, Anzahl Ergebnisse)
  und 10 beste Ergebnisse
- Volle Ergebnisse: `/$queryid/page_$nr.json`

Optimierungen

- Ziel: auch größte Suchanfragen unterstützen
- Ziel: möglichst schnell Antworten liefern
  angenehmer für den Nutzer :)
- Tools: Monitoring + pprof zum Profiling

Optimierungen: vorher/nachher



optimierte Version deployed

Optimierung (1): RAM

- Proof of concept: alle Ergebnisse im Speicher
  → Out of memory (OOM) bei großen Anfragen

- Ergebnisse auf SSD schreiben, external sorting:
  → (ranking, offset) tuples im Speicher halten
  → nach `ranking` sortieren
  → ab `offset` lesen

Optimierung (2): > 70 Byte pro Ergebnis

```
type resultPointer struct {
    backendidx  int
    ranking     float32
    offset      int64
    length      int64
    path        string
    packageName string
}
```

Optimierung (3)

- `sync.Pool` um Paketnamen nur einmal im Speicher zu halten (`string` → `*string`)

- `hash/fnv` über `path` speichern, kann immernoch als Tiebreaker genutzt werden (`string` → `uint64`)

Optimierung (4): 36 Byte pro Ergebnis

```
type resultPointer struct {
    backendidx  int
    ranking     float32
    offset      int64
    pathHash    uint64
    packageName *string
}
```

Optimierung (5): CPU

- `page_$page.json` on-demand generieren, spart Speicherplatz und ca. 30 Sekunden sort/read/write

- capnproto statt `encoding/json` nutzen, bei signifikanten Mengen (>> 1 MB/s) relevant

Optimierung (6): syscall overhead

- Mitzählen statt `lseek(2)` (syscall-overhead)

- `bufio` nutzen, 250× weniger `read(2)` syscalls beim Deserialisieren

Ausblick

- Meisten Suchen keine Regexp, sondern Fehlermeldungen und Identifier (Funktionen)
  ca. 74% Identifier, 26% Regexp

- Suffix Arrays würden Identifier beschleunigen
  Nachteil: teuer (Disk, CPU), könnte aber gehen

Ende. Fragen?

- Blog mit Beiträgen zu DCS:
  https://people.debian.org/~stapelberg/

- Bitte gebt mir Feedback zum Vortrag!
  http://goo.gl/forms/xDcQrTnwpg