

Performance mit high-traffic TYPO3 sites

Christian Kuhn
e-netconsulting KG
<lolli@schwarzbu.ch>
follow @lolli42

Inhalt

- Kurzer Überblick kress.de
- Varnish als Reverse Proxy
- Einführung caching framework und page cache
- Nutzung des caching frameworks auf content element Ebene mit enetcache
- Fragen / Demo / Code

Überblick kress.de (1)

- Bis zu 30 Plugins (Home)
- Drei große Seitenbereiche:
 - ~120.000 News, unterteilt in Kategorien
 - ~40.000 "Köpfe" mit unterschiedlichen Querverbindungen zu anderen Seitenbereichen
 - Mehrere tausend Spots mit Zusatzinformationen
- Newsletterversand innerhalb weniger Minuten an mehrere zehntausend Empfänger, starker Zugriffsspeak

Überblick kress.de (2)

- Diverse "Hotlisten": meistgeklickt, Top / Flop-rating, ...
- Volltextsuche über alle wichtigen Bereiche
- Hosting mit nur einem Server:
 - 12GB RAM
 - Quad-Core i7 mit HT
 - Keine Spezialhardware wie SD-Platte für mysql o.ä.
- Parsing der ungecachten Homepage: Fast drei Sekunden

Varnish – Statische Requests

- Moderner Proxy, squid überlegen (http only)
- Extrem schnell, erzeugt sehr wenig Rechenlast, RAM driven
- Beste Messung: ~15k Zugriffe über loopback Interface ohne weitere Optimierungen
- Relativ einfach zu konfigurieren für statische Dateien, far-future expire headers per .htaccess für typo3temp, uploads, ...
- 95% aller statischen Zugriffe nach wenigen Minuten durch varnish abgefangen

Varnish – Dynamische Requests

- Twitter und mail Links erhalten Spezialpfad als preVars in realurl, die sonst ignoriert / wieder gelöscht werden
- Varnish reagiert auf mail/ und tweet/ und cacht diese HTML-Seiten für kurze Zeit
- In Produktion: >1k Hits pro Sekunde in varnish nach Newsletterversand, <50 Hits / s apache, varnish RAM Peak < 600MB
- CPU Last steigt nur unwesentlich, Load um 1 bis 2

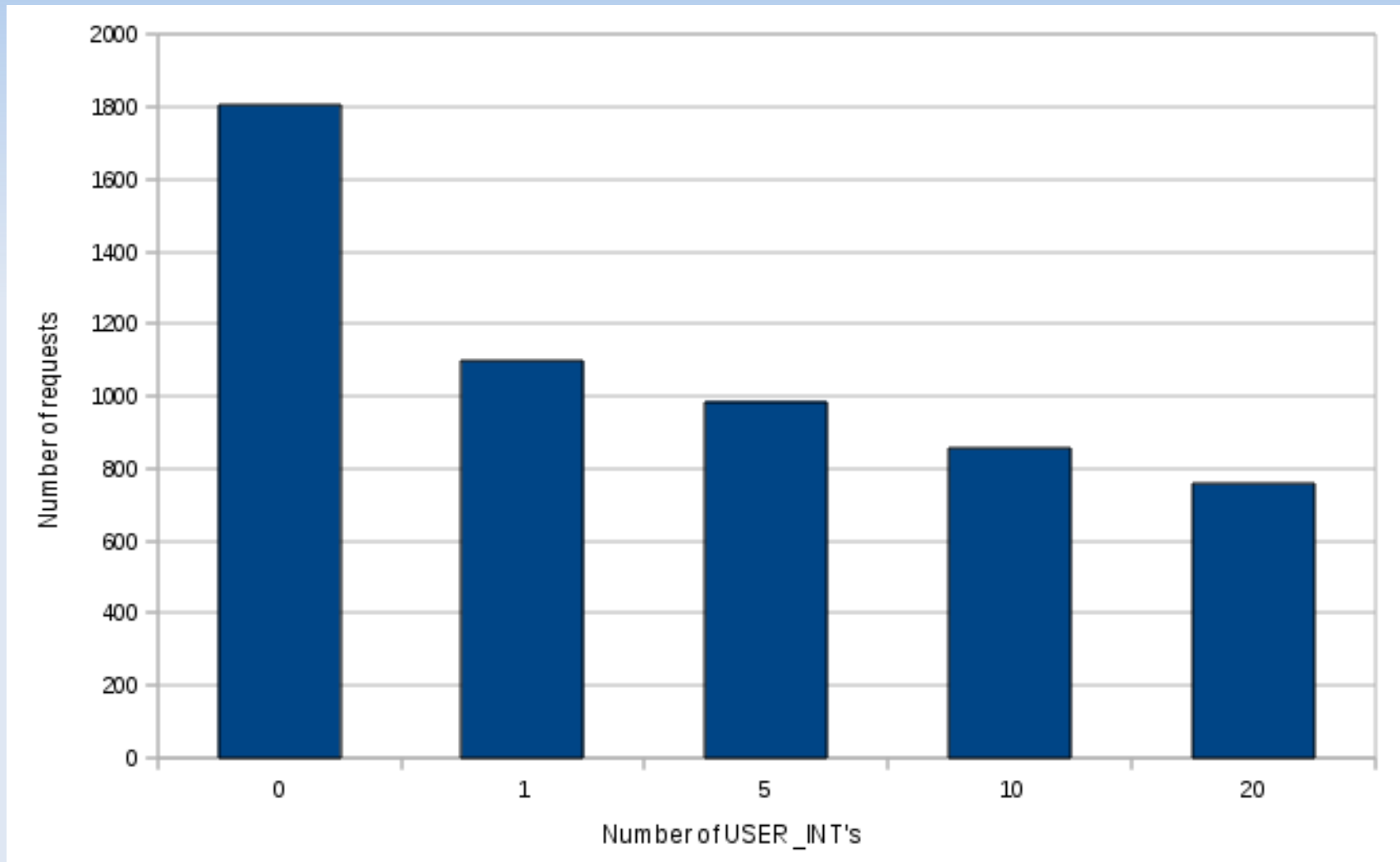
Varnish – Ausblick

- Daemon stabil, bisher ein Restart aus unerfindlichen Gründen
- Debian Version relativ alt, möglicherweise selbstkompilierte Version im nächsten Projekt
- Syntaxänderungen in v2, daher sind Konfigbeispiele teilweise falsch
- v2: Dirty-Cache delivery :)
- Future: Gezieltes Füttern / Invalidieren von Cacheeinträgen aus TYPO3?

Caching in TYPO3

- Gut konfigurierter Server kann eine voll gecachte Seite in 20-30ms ausliefern
- Erstes USER_INT auf einer Seite verdreifacht die Parsetime: ~80ms mit hello world pi1
- Weitere USER_INT's erhöhen die Parsetime nicht mehr so stark
- Problem bei kress: Entweder viele USER_INT's wegen vieler Hotlisten, oder niedrige lifetime.
- Entscheidung: Eigenes caching System, persönliche Informationen per eld-Skript

Performance impact USER_INT



Vermeidung von USER_INT's

- Einige der Hotlisten bei kress sind recht aufwändig, trotz optimierter Queries benötigt die Datenaufbereitung teils 100ms und mehr
- Bei bis zu 5 Sekunden Renderzeit für die aufwändigsten Seiten "verhungern" einige Requests, Requests auf die gleiche Page werden geblockt, ständige "Aussetzer" wichtiger Seiten bei Verzicht auf USER_INT, dafür niedrige cache lifetime.

Caching Framework – Basics (1)

- Backport aus flow3
- Kann unterschiedliche Datenbackends anbinden (memcache, apc, db, file, ...)
- Ein Eintrag besteht aus:
 - Lifetime in Sekunden
 - Unique identifier, der schnell berechnet werden kann
 - 0-n Tags
 - Data

Caching Framework – Basics (2)

- Initialisierung des Frameworks dauert 1-2ms länger als das herkömmliche Caching
- Page cache: Tagging mit id der Seite (pageId_4711)
- Problem (besonders tt_news): Tagsüber werden permanent News bearbeitet, Einsatz von TCEMAIN.clearCacheCmd würde ständig große Teile des Caches leeren

Caching Framework Performance

- Memcached + apc Backend sehr langsam, besonders beim Löschen + Hinzufügen von Rows
- Langsames memcache Architektur bedingt (90% php CPU Last, 10% memcache Last), Helfer needed!
- Einsatz des db Backend empfohlen, file Backend sollte ebenfalls recht fix sein (Nadelöhr: Schreiblast), ggf. Test mit RAMDisk?!

Enetcache – Frontend

- Caching auf plugin content Element Ebene
- Frontend Plugins holen und setzen Einträge
- Tagging der Einträge mit benutzten Datensatzrows (tt_news_4711)
- Einzelne lifetimes für jedes content Element
- Tagging des page cache Eintrags mit allen Tags aller Plugins einer Seite, niedrigste lifetime
- Wiederverwendung von cache Einträgen

Enetcache – Backend

- Tag dropping in enetcache + page cache mittels Hooks in TCEmain, anhand Tabellename, Aktion, ID
- Forward dropping: News Kategorieseiten werden mit Kategorie-ID getagt, TCEmain hook sucht nach vorwärts Referenzen im bearbeiteten Datensatz und dropt cache Einträge mit zugeordneter Kategorie
- Tag dropping leicht erweiterbar (zB. ext:comments)

Enetcache - Goodies

- Scheduler Task zur garbage collection von alten cache Einträgen
- Fertige, transparente Plugin Wrapper für bestehende Extensions
- Core patches für einige Nadellöcher im caching framework
- Umfangreiche Dokumentation
- Leichte Erweiterbarkeit, nutzbare Interfaces für Dispatcher Klassen

Enetcache - Debugging

- Enetcacheanalytics hookt in enetcache und tslib_content, bereitet cache Aktionen grafisch auf
- Logging aller enetcache Aktionen, inklusive Tag dropping im BE, droppen einzelner Cache rows
- Zeigt alle relevanten Informationen für einzelne Seiten:
 - Anzahl der Plugins, mögliche Ersparnis
 - Identifier, Tags, Lifetime
 - Cache hits / cache misses

enetcacheanalytics

Cache analyzer | Youngest log entry | Show only page id | Refresh

2010-02-14 18:39:11: Display request with unique ID 4b78353f430d5.
 20 of 47 rendered plugins used enetcache (42.55%).
 19 of 20 enetcache get requests where successful (95.00%).
 Could have saved 20ms of 431ms (4.64%).

PID	UID	FE - BE User	Life time / Valid until	Caller	Identifier	Tags	Data	Time spent
67	68	0 - 49		class: tx_enetdoubleclick_pi1 function: main				4
67	68	0 - 49		class: tx_enetdoubleclick_pi1 function: main				2
67	68	0 - 49		class: tx_kresstimeline_pi1 function: main				3
68	785	0 - 49		class: tx_kressnews_dispatcher function: main				32
		0 - 49		function: main class: tx_kressnews_dispatcher	32d7ee12d556c531695b326f61e908c8	b:0;		
		0 - 49	2010-02-15 18:39:15 = +86400	function: main class: tx_kressnews_dispatcher	32d7ee12d556c531695b326f61e908c8	tx_kressnews_controller_homepage tt_news_101948 tt_news_101948 32d7ee12d556c531695b326f61e908c8		20
68	1099	0 - 49		class: tx_kressnews_dispatcher function: main				8
		0 - 49	2010-02-15 18:38:30 = +86355	function: main class: tx_kressnews_dispatcher	d3baab112a5f9357dcba0280ddd7ac60	tx_kressnews_controller_homepage tt_news_101936 tt_news_101946 tt_news_101747 tt_news_101709 tt_news_101938 tt_news_101715 tt_news_101936 tt_news_101946 tt_news_101747 tt_news_101709 tt_news_101938 tt_news_101715 d3baab112a5f9357dcba0280ddd7ac60		

Enetcache – Use case Homepage

- 2,5 – 3 Sekunden Renderzeit ohne enetcache
- ~700ms mit enetcache, typischerweise wird genau ein Plugin neu berechnet
- Wegen geringer Lifetime keine Tags für die Hotlisten
- Lifetime der News Einträge 24h oder mehr
- Nur wenige Bugs nach Lifegang, Redaktuere können und müssen Cache manuell nicht leeren

Erfahrungen in Produktion (1)

- Page cache wird 4GB und größer, innodb cache entsprechend angepasst
- Später: Umstellung auf gzip compressed db Backend, verkleinert page cache Tabelle um 70%
- Enetcache Tabelle < 1GB, ohne gzip
- Mysql db RAM driven, diverse Optimierungen (besonders innodb) innerhalb der ersten Woche, danach keine slow Queries mehr

Erfahrungen in Produktion (2)

- Viele slow INSERT's im mysql Log, daher `t3lib_db exec_INSERTmultipleRows()` (TYPO3 4.4)
- "DELETE FROM cacheTable WHERE SELECT identifier FROM tagsTable" extrem langsam da mysql keine Indexes für die outer Query nutzt, Splitten in einzelne Queries ergibt Faktor 1000.
- ToDo: Entfernen des `auto_increment` Feldes beschleunigt INSERT, DELETE und SELECT etwas

Enetcache - Probleme

- Wenn Plugins TSFE ändern (zB. AdditionalHeaderData) müssen diese Informationen parallel zu den Daten gespeichert werden in enetcache und nach successful get() wieder gesetzt werden
- USER_INT's innerhalb von gecachten USER Objekten schwierig, da Informationen in TSFE verloren gehen
- Kein automatischen start- / endtime handling bei Datensätzen, kein clearAtMidnight

Enetcache - Ausblick

- Mehr Plugin Wrapper für bestehende Extensions (besonders tt_news v3)
- Core hacks zurück nach Upstream:
 - Garbage collection task
 - Performance Verbesserungen im Framework
 - Diverse Indexes in anderen Tabellen
- Möglicherweise weitere Backends testen (zB. LDAP)

Diverses

- Monitoring apache, varnish, mysql
- Feinoptimierung mysql dringend erforderlich für annehmbare Performance
- Vorgezippte, zusammengefügte .js und .css Files
- Far-future expire Headers für alle statischen Dateien
- mod_deflate, niedriges keep alive in apache
- Ersatz von apache durch nginx mit fcgi?

Aufwand

- In Summe haben alle caching und performance Optimierungen bis zu 20% des gesamten Projektaufwandes verschlungen, in weiteren Projekten weniger
- Entwickler konnten enetcache nach relativ kurzer Zeit sauber einbinden, wenig bis keine kritischen Bugs
- Gut 2 Tage Aufwand mit zwei Entwicklern um alle Tags und Identifier abzuklopfen und zu prüfen

Fazit

- Varnish kann mit einfacher Konfig bereits einen Großteil der Requests auf den langsamen apache abfangen
- 3-Level-caching (varnish, page-cache, enetcache) erlaubt Fehler in einer Ebene ohne Zusammenbruch des gesamten Systems
- Enetcache ist besonders bei dynamischen Seiten mit vielen Änderungen sinnvoll
- Systemüberwachung und Tuning mindestens in der Anlaufphase

Fragen, Code-examples, Demos

Christian Kuhn
e-netconsulting KG
<lolli@schwarzbu.ch>
follow @lolli42