Technische Universität München
WWW & Online Services

Performance grosser Plone Installationen

Cave Ats und Optimierungsmöglichkeiten

Dr. Thomas Wagner

Charakter großer Zope/Plone Installationen

- hohe Zugriffszahlen
- grosse Objektzahl
- ZEO mit mehreren Frontends
- Reverse Proxy (z.B. Squid)
- Einbindung externer Services

und gegebenenfalls

- hoher Anteil an Schreiboperationen
- personalisierte Informationen

Allgemeine Seitengrösse

Die ausgelieferte HTML-Seiten von Plone sind generell sehr gross, unabhängig vom Inhalt:

- umfangreiche Decoration
- Barrierefreiheit
- Javascripts
- Styles

Abhilfe

- Decoration reduzieren
- Styles und Scripts differenziert einbinden

Dynamische Komponenten

Ein Einsatz dynamischer Komponenten umgeht ein mögliches Caching durch den Proxy und verbraucht Renderzeit:

- Stlyesheets werden oft per DTML generiert
- Einsatz von dynamischen Komponenten auch an Stellen wo dies nicht notwendig wäre
- suboptimaler Code (Scripts, ZPT, ...)
- Performance Leaks in Skintemplates und häufig aufgerufenen Seiten (Objecthandling, <tal:...>

Abhilfe:

- Stylesheets einmalig generieren und statisch einbinden
- Code-Optimierung
- Performance-Killer aus Skins entfernen

Personalisierung

Für Personalisierung sind umfangreiche Conditions, Queries etc. notwendig. Diese kosten Rechenzeit

- individuelle Portletinhalte
- benutzerspezifische Abfragen für Content
- Conditions und Auswertung von Properties
- allgemein notwendiges Handling von Objekten

Abhilfe:

- nur dort Personalisierung einsetzen, wo notwendig
- nur diejenigen Seite personalisieren bei denen Personalisierung sinnvoll ist (z.B. keine Portlets die für alle Bereiche gelten)
- Queries und Conditions optimieren (einmalig bei Login durchführen, nur auf Flag testen)

Objects versus Brains

Handling von echten Objekten in ZPT extrem inperformant, insbesondere bei grossen Objektzahlen und grossen Objekten (Objekte mit vielen Attributen, Unterobjekten)

- z.B. Verwendung von objectValues bei Auflistung von Folderinhalten, Batch-Darstellung
- Schleifen/Conditions über Objects statt Brains
- usw.

Abhilfe:

- Nutzung von Brains und Brain-Metadaten wo immer möglich
- Objekte nur dann holen, wenn notwendig

Grosse Dateien

Das Handling von grossen Dateien wie Images und Files ist inperformant.

- hohe Ladezeiten der Objekte

Abhilfe:

- Cachen durch Proxy (Squid)
- ZODB 3.8 ?

<tal:block>, <metal:block> ...

Die Verwendung gestaffelter <tal:..> und <metal:...>-Blöcke in Skins führt unter hoher Request-Last zu massiven Performance-Einrüchen

- Problem noch nicht vollständig analysiert: Parallelisierung bei hoher Requestzahl innerhalb einzelner Methoden im Publisher und anderswo führt zu Lock
- Wahrscheinlichkeit für Lock steigt mit Request-Häufigkeit und Anzahl der <tal:...>, <metal:...> im Code
- Lock-Zeiten sind sporadisch variieren sehr stark

Abhilfe:

- statt <tal:...>, <metal:...> wo es möglich ist echte Tags verwenden oder <span tal:omit-tag>

ZODB Cache und Arbeitsspeicher

Object-Cache der Datenbank muss ausreichend gross sein und mit dem verfügbaren Arbeitsspeicher der Frontends korrespondieren:

- bei hoher Objektzahl hoher Speicherbedarf im RAM der Frontends
- Erreichen der Swap-Grenze unbedingt vermeiden

Abhilfe:

- maximale System-Process-Size erhöhen
- Frontends mit maximalem RAM ausstatten (ggf. 64Bit System)
- Automatischer Restart der Frontends bei Erreichen der Speichergrenze (optimalen Wert empirisch ermitteln)

ZEO/Client Cache Invalidierung

Häufiger ZEO Zugriff z.B. durch eine hohe Invalidierungsrate eines zu kleinen Client Caches oder wegen Restart des Frontends führt zu Engpässen bei ZEO bzw. Netz

- zu geringe Cachesize insbesondere bei Operationen mit vielen Objekten (ohne Hit) problematisch (z.B. Catalog Query mit getObject)
- korrupte Cache Files veranlassen häufige Invalidierung
- Gleichzeitiger Neustart mehrerer Frontends

Abhilfe:

- Client Cache gross genug dimensionieren
- Cache Konsistenz prüfen bzw. Caches gelegentlich purgen
- Restarts und Cache Purging der einzelnen Frontends immer staffeln, nie gleichzeitig

Prozess-Parallelisierung

In einigen Fällen kommt es zu einer Parallelisierung von Teilprozessen (nicht Requests). Dies führt bei hoher Request-Last zu Locks (siehe <tal:...>, <metal:...>

- Phänomen bisher nicht detailiert analysiert
- Durch Lock auch alle anderen Requests betroffen

Abhilfe:

- Vermeidung kritischen Codes (bisher vorallem <tal:...>, <metal:...>
- Ausreichend schnelle Frontends, so dass Requests hinreichend schnell abgearbeitet werden um Parallelisierung zu vermeiden

Catalog Nutzung

Plone macht intensiven Gebrauch vom portal_catalog. Durch eine hohe Zahl gleichzeitiger Zugriffe kommt es dadurch zu Performance-Engpässen

- Read Conflict Errors bei gleichzeitigen Schreib-und Lesezugriffen unvermeidbar. Schreib- und z.T. auch Lesevorgänge dadurch stark verzögert.

Abhilfe:

- QueueCatalog: jedoch nur bedingt einsetzbar, wenn nicht alle Informationen des Objects sofort verfügbar sein müssen und ein Zeitslot mit geringem Catalo-Zugriff verfügbar ist.
- Für spezielle Anwendungen spezifische ZCatalogs verwenden (portal_catalog nur für Portal-weite Daten und Objekte)

Read Conflicts allgemein

Bei hohen Zugriffszahlen auf Objekte und gleichzeitigem Schreiben treten Read Conflicts auf. Die Auslieferung des Objekts wird verzögert, bis Konflikt beseitigt ist

- Session
- Portal Catalog
- Erstellen von Subobjekten in häufig angefragten Objekten

Abhilfe:

- Session unabhängig von ZODB machen (mySQL Session)
- QueueCatalog sowie spezielle Sub-Kataloge
- BTree Folder verwenden

Externe Services

Die direkte Einbindung externer Services (RSS, XML-RPC, SOAP, LDAP, mySQL ...) ist grundsätzlich problematisch. Der Aufruf ist zeitintensiv, nicht beeinflussbar und kann zu Deadlock des Portals führen, wenn mehr Requests ankommen als abgebaut werden

- hoher Overhead (Parser, Envelope, Autentifizierung, Handschakes)
- Verfügbarkeit und Performance des Remote-Systems

Abhilfe:

- direkter Einbau nur in Seiten die selten bzw. spezifisch aufgerufen werden (also nicht in Portlets oder Templates die praktisch überall benutzt werden, RSS!)
- Entkopplung der Portal-Requests von den Request beim Service durch geeignete Tools

Ganz speziell: timeoutsocket.py

- Achtung bei Benutzung des socket timeouts: Dieser wird ggf. global modifiziert und hat damit Einfluss auf alle Zugriffe auf externe Systeme (RSS, LDAP, mySQL usw.)
- einzelne timeoutsocket-Module fehlerhaft: sporadische nicht nachvollziehbare Hänger bei allen externen Zugriffen des Portals auf andere Systeme (LDAP, IMAP ...)

Fazit

- Plone und Zope bieten eine ganze Reihe von Performance-Fallen, die erst bei grösseren Installationen bzw. bei entsprechender Last auftreten und bei Testinstallationen oder unter normalen Bedingungen nicht auffallen.
- Viele Probleme liegen bereits im Systemaufbau aber auch in Skin und Code
- Ein Caching von Inhalten ist erst sekundär relevant