Anforderungs-Spezifikation für den Webstart-Proxy

Ist-Zustand

Der bestehende Webstart-Proxy ist ein wenig in die Jahre gekommen. Vom Source-Level ist er eher Java1.4, vom Codestyle erinnert er an COBOL :-)

Des weiteren ist die Codebasis durch niemals verwendete Features wie JarDiff endlos aufgebläht. Der Speicherverbrazch und das IO-Verhalten sind auch nicht unbedingt optimal.

Des weiteren ist es erwähnenswert, dass der Proxy nicht mehr auf dedizierten Server-Maschinen läuft, sondern "nebenbei" auf normalen Client-Rechnern. Auf diesen Rechnern wird also gleichzeitig ganz normal gearbeitet.

Anforderungen an die neue Implementierung

1. IO-Verhalten

Die aktuelle Proxy-Implementierung geht bei wenigen 100 parallelen Verbindungen "in die Knie". Dies ist dem verwendeten Classic-IO-Modell geschuldet.

Die Neu-Implementierung soll mit einer (theoretisch) unbegrenzten Anzahl von parallelen Requests umgehen können ohne die OS-Resourcen über Gebühr zu belasten. Das funktioniert natürlich nur mit async IO, aka "NIO".

Bezüglich IO sind zwei Subsysteme zu betrachten:

- Frontend
- Backend

Das Frontend stellt die Schnittstelle zu den Clients dar. Hier sind tausende parallele Requests zu erwarten.

Das Backend stellt die Verbindung zum Parent-Proxy bzw zum Download-Portal dar. Hier muss die Anzahl von parallelen Verbindungen limitiert werden.

1.1 WriteThrough-IO

Eine Resource, welche im Cache nicht gefunden wird, wird vom Backend herunter geladen. Dieser Download kann natürlich einige Zeit dauern, die Clients sollen wärend dieser Zeit aber nicht blockieren.

Es ist also ein Mechanismus zu schaffen, um beim Backend-Download bereits empfangene Daten an die Clients ausliefern zu können. Dabei muss sicher gestellt werden, dass eine Resource nur einmal vom Backend geladen wird, gleichzeitig aber an viele Clients ausgeliefert werden kann.

2. Speicher-Verhalten / CPU-Benutzung

Es ist zwingend auf Speicher-Sparsamkeit und CPU-freundliches Verhalten zu achten. Die Verwendung von separaten Empfangs-Buffern pro Verbindung ist so ein Thema.

"CPU-Freundlich" bedeutet hier, dass nicht beliebig viele Threads erzeugt werden sollen. Das klassische IO-Modell mit einem Thread pro Verbindung kommt also nicht in Frage. Der komplette Front-End-Traffik wird also über EINEN Thread behandelt.

Der Backend-Traffik verwendet eine begrenzte Anzahl von Verbindungen. Hier kann "ClassicIO" verwendet werden und pro Connection ein Thread. Das macht das ganze deutlich einfacher, lässt sich aber auch noch auf NIO umstellen.

Caching

Bereits herunter geladene Resourcen sind in einem Disk-Cache zu verwalten. Für die Auslieferung der Resource werden außer dem Content noch Meta-Informationen benötigt. Das ist zum Beispiel der Content-Type und die Länge der Resource.

Der Cache muss in der Lage sein, diese Meta-Informationen zusammen mit dem Content zu verwalten.