WebLicht: Bombardieren bevor die Services explodieren

Daniël de Kok, Wei Qiu, Marie Hinrichs

Einleitung

Web-Analyse Software, die Informationen über das Nutzerverhalten sammelt und auswertet, kann eingesetzt werden, um Probleme zu verhindern, bevor diese entstehen. In diesem Abstrakt zeigen wir, wie wir eine derartige Software auf ein System mit verteilten Webservices, WebLicht [1], angewandt haben und wie wir die entstandene Analyse benutzt haben, um das System zu verbessern.

WebLicht ist eine Webanwendung zur automatischen Annotation von Texten und multimodalen Daten. WebLicht nutzt eine serviceorientierte Architektur. Dies ermöglicht den CLARIN-Zentren Annotationswerkzeuge hinzuzufügen durch diese als Webservice zu implementieren und die CMDI-Metadaten [2] über diesen Service an ihr Repositorium zuzufügen. WebLicht aggregiert diese Metadaten und bietet dem Nutzer die Annotationswerkzeuge an. Wenn ein Benutzer mehrere Annotationswerkzeuge ausführen will oder ein Annotationswerkzeug vom Output eines anderen Werkzeugs abhängig ist, erlaubt WebLichts Verkettungsmechanismus es den Benutzern, Annotationswerkzeuge auf eine kompatible Art und Weise zu kombinieren.

WebLicht ist eine voll entwickelte Anwendung, die mehr als hundert Annotationswerkzeuge anbietet. Dadurch weitet sich die Nutzung von WebLicht in zwei Dimensionen aus: (1) Die Anzahl der WebLicht-Benutzer steigt; (2) durchschnittlich lassen die Benutzer größere Datasets annotieren. Um diese Wachstumsstrukturen zu verstehen, sammeln wir Nutzungsstatistiken. Um WebLicht und die Annotationswerkzeuge für unsere Nutzerbasis zu optimieren, müssen wir zuerst die Nutzungsmuster der derzeitigen Benutzer kennen. Zweitens müssen wir zukünftige Kapazitätsengpässe vorhersagen können, damit wir uns mit ihnen auseinandersetzen können, bevor sie die User Experience beeinträchtigen.

In den folgenden Abschnitten werden wir zuerst beschreiben, wie wir Benutzeraktivität und Nutzungsmuster messen. Dann werden wir die Simulation verschiedener Nutzungsszenarien behandeln. Schließlich werden wir einen kurzen Überblick über die Veränderungen geben, die wir im vergangenen Jahr aufgrund der Messungen vorgenommen haben.

Die Messung von Benutzeraktivität

Das Sammeln von Nutzerstatistiken für Webseiten ist ein wohlverstandenes Feld, in dem es mehrere konkurrierende und voll entwickelte Produkte, wie etwa Google Analytics, gibt. Es sind auch viele gute Open Source Webanalyse-Tools, wie zum Beispiel Piwik[1], Webalizer[2] and AWStats[3] verfügbar. Diese Tools funktionieren gewöhnlich auf eine von zwei Arten: (1) Sie analysieren die vom Webserver protokollierten Logdateien; oder (2) sie benötigen einen Maintainer, der auf jede Seite einen Javascript-Snippet einfügt, wodurch der Browser des Benutzers mit der Analysesoftware Kontakt aufnimmt.

Bereits existierende Lösungen sind nicht direkt auf WebLicht anwendbar, da ihre Verwendung nur zeigen würde, wie oft, aus welchem Land, etc. WebLicht besucht wurde. Die nützlichste Information würden fehlen: welche Annotationswerkzeuge verwenden die Benutzer, und wie oft? Piwik bietet eine Programmierschnittstelle (Application Programming Interface) an, durch die jedes Annotationswerkzeug seine eigene Nutzung registrieren kann. Jedoch würde dies erfordern, dass knapp hundert Dienste upgedatet werden müssten, um die Programmierschnittstelle aufrufen zu können. Ein weiterer Nachteil dieser Lösung ist, dass interessante Informationen, wie etwa das Land des Benutzers, nicht verfügbar sind, da die Annotationswerkzeuge von WebLicht aufgerufen werden und nicht direkt vom Browser des Benutzers.

Wir haben die oben genannten Probleme durch das Melden von Statistiken im WebLicht-Verkettungsmechanismus gelöst. Da der Verkettungsmechanismus jedes Annotationswerkzeug aufruft, kann er gleichzeitig die Verwendung des Werkzeugs über die Piwik-Programmierschnittstelle melden. Dies macht die Statistiken sofort erhältlich für alle Annotationswerkzeuge, die in WebLicht verfügbar sind, ohne auch nur eine von ihnen zu verändern. Da der Verkettungsmechanismus in der WebLicht-Anwendung ausgeführt wird, hat es den Zusatznutzen, dass wir einige Metadaten bereitstellen können, die es Piwik erlauben das Land des Besuchers, den Webbrowser, etc. zu ermitteln.

[1] http://piwik.org/
[2] http://www.webalizer.org/
[3] http://www.awstats.org/

⊞ **ws1-clarind.esc.rzg.mpg.de**	12	13
⊞ **weblicht.sfs.uni-tuebingen.de**	372	18026
⊞ **kaskade.dwds.de**	12	12
⊞ **dspin.dwds.de:8080**	51	64
⊞ **dlexdb.de**	5	5
⊟ **clarin05.ims.uni-stuttgart.de**	139	3832
/treetagger2008	35	55
/treetagger	4	4
/RFTaggerMorph	26	3645
/rftagger	1	1
/cgi-bin/dspin/tokeniser4.perl	43	75
/cgi-bin/dspin/tei2tcf4.perl	1	1
/cgi-bin/dspin/tei2tcf3.perl	2	3
/cgi-bin/dspin/smor4.perl	1	1
/cgi-bin/dspin/bitpar4.perl	26	47
⊞ **chopin.ipipan.waw.pl:8083**	1	1

Abb. 1: Piwik Nutzerstatistiken für die Stuttgarter Werkzeuge

Die Simulation von Nutzungsmustern

Eines der Probleme, auf das wir mit der gestiegenen Nutzung von WebLicht gestoßen sind, ist, dass manche Annotationswerkzeuge nicht zur Bewältigung vieler Simultanbenutzer oder großen Inputs entwickelt wurden. Leider wurden diese Probleme oft erst entdeckt, wenn ein Benutzer eines der Annotationswerkzeuge nicht ausführen konnte.

Durch das Simulieren von WebLicht-Nutzungsmustern konnten wir solche Probleme aufdecken, bevor die Benutzer ihnen begegnen. Zu diesem Zweck haben wir ein Simulationswerkzeug namens Bombard entwickelt. Bombard ermöglicht es den Entwicklern von WebLicht-Annotationswerkzeugen, Testfälle zu spezifizieren. Jeder Testfall besteht aus: der Kette von Annotationswerkzeugen, die getestet werden soll; dem Input; einem Intervall, das anzeigt, wie oft der Testfall ausgeführt werden soll; und die maximal erlaubte Bearbeitungsdauer. Eine Bombard-Konfigurierung kann aus vielen solcher Testfälle bestehen. Wenn Bombard gestartet wird, beginnt es jeden Testfall zu einem willkürlich gewählten Zeitpunkt und wiederholt den Testfall ab da in dem festgelegten Intervall. Während des 'Bombardements' behält Bombard den Überblick über Fehlschläge und inakzeptable Bearbeitungsdauern. Danach können die Entwickler einen Bericht mit Statistiken für jedes Annotationswerkzeug abrufen.

Im CLARIN-Zentrum in Tübingen wird Bombard zum Testen neuer Dienste genutzt, indem es das folgende Szenario simuliert: Zwei Gruppen von jeweils 40 Studierenden reichen innerhalb von zwei Minuten einen Text ein. In Gruppe A bearbeitet jeder Studierende zwei Textabschnitte aus Wikipedia, in Gruppe B bearbeitet jeder Studierende den Roman *Alice im Wunderland* (oder eine Übersetzung davon). Mithilfe dieses Testszenarios konnten wir Kapazitätsengpässe

in früher entwickelten Annotationsdiensten bestimmen und sicherstellen, dass die neuen Dienste in der Lage sind, solche Szenarien zu verarbeiten.

Da die Konfigurierung der Testfälle in Bombard anpassungsfähig ist, kann sie einfach auf andere Szenarien oder Annotationswerkzeuge, für die andere Erwartungen gelten, angewendet werden.

Änderungen an WebLicht

Wir haben unser Klassenzimmer-Szenario an Annotationsketten getestet, die unseren Feststellungen nach häufig genutzt werden, nämlich: Part-of-speech Tagging, Lemmatisierung, Konstituenz-Parsen, Dependenz-Parsen und Named-entity Recognition. Während der Simulation entdeckten wir die folgenden Probleme: (1) Manche Dienste versagten, wenn viele Instanzen des längeren Textes gesandt wurden; (2) manche versagten an dem längeren Text, da er relativ verrauscht ist; (3) bei manchen Diensten, insbesondere Parsern, kamen die Anfragen schneller herein als der Dienst sie verarbeiten konnte.

Das erste Problem war am leichtesten zu lösen – diese Dienste hatten eine ältere Version der TCF interchange format library genutzt, die sich nicht linear an die Größe des Inputs anpasste. Das zweite Problem musste von Fall zu Fall einzeln gelöst werden. Diese Dienste hatten einige Programmierfehler, die sie bei unerwartetem Input versagen ließen. Das dritte Problem jedoch war schwerer zu lösen und wird im Rest dieses Abschnitts besprochen werden.

Parser für natürliche Sprachen sind oft langsam im Vergleich zu anderen Annotationsdiensten. Zum Beispiel können gebräuchliche Konstituenz-Parser normalerweise höchstens ein paar Sätze pro Sekunde parsen. Um die von uns vorhergesehenen Gebrauchsfälle bewältigen zu können, mussten wir die Fähigkeiten moderner Server und Computercluster, Prozesse parallel durchführen zu können, ausnutzen. Dafür haben wir ein Framework entwickelt [3], das auf einer verteilten Task-Warteschlange (Jesque) basiert und das folgende bietet: Parallele Prozessverarbeitung innerhalb der Anfragen, gleichzeitige Verarbeitung von Anfragen, Garantien, was die Verwendung von Ressourcen und Fairness betrifft (z.B.: Eine große Anfrage sollte keine sichtbaren Auswirkungen auf kleine Anfragen haben.)

Wir benutzen dieses Framework in den upgedateten Diensten für die Malt-, Stanford- und Berkeley-Parser. Dadurch können wir solche Szenarien leicht bewältigen. Was vielleicht noch wichtiger ist: Unser Framework erlaubt es uns, Webservices an noch größere simultane Benutzerzahlen oder größere Inputs anzupassen, indem wir weitere Prozessorkerne oder Geräte hinzufügen.

Fazit

Wir haben diesem Abstract zwei neue Vorgehensweisen zur Messung der Nutzung und Kapazität von WebLicht dargestellt. Zuerst haben wir Nutzungsmeldungen zum Verkettungswerkzeug hinzugefügt, sodass wir die Nutzungsstatistiken der Annotationswerkzeuge bekommen ohne die CLARIN-Partner darum bitten zu müssen, ihre Werkzeuge anzupassen. Zweitens haben wir das Dienstprogramm Bombard vorgestellt, das es uns ermöglicht die Auswirkung der Hochrechnung des momentanen Wachstums zu messen. Solche Messungen machen uns die Nutzung zur Bestimmung von Kapazitätsengpässen in der WebLicht-Infrastruktur und ihre frühe Behandlung möglich.

Bibliografie

[1] Hinrichs, E., Hinrichs, M., and Zastrow, T. (2010). WebLicht: Web-based LRT services for German. In *Proceedings of the ACL 2010 System Demonstrations*, pages 25– 29. Association for Computational Linguistics.

[2] ISO 24622-1:2014. 2014. Language resource management -- Component Metadata Infrastructure (CMDI) -- Part 1: The Component Metadata Model. Technical Report, ISO.

[3] De Kok, D., De Kok, D., and Hinrichs, M. (2014). Build your own treebank. In: *Proceedings of the CLARIN Annual Conference 2014.* Soesterberg, Netherlands.