合肥工業大學

英文翻译

（翻译中文）

译文题目 通过 WebSocket 异步发布订阅框架建立实时 Web 应用程序

学生姓名 李磊

学　　号 2013214381

专业班级 软件 13-2 班

2017 年 6 月 1 日

通过 WebSocket 异步发布/订阅架构建立实时 Web 应用程序

作者：Jastian Ganaputra：印度尼西亚雅加达 Bina Nusantara 大学信息技术的研究生

Bens Pardamean：印度尼西亚雅加达 Bina Nusantara 大学计算机科学副教授，生物信息与数据科学研究中心主任

摘要：本研究应用了发布/订阅架构模式，使用 Amazon Simple Queue Service 作为消息代理，以便于以传输层的 WebSocket 协议为特色的 HTML5 上的发布/订阅架构。 在 WebSocket 连接上应用发布/订阅模式解决了 WebSocket 连接上丢失消息的问题。 将框架性能与 WebSocket 连接本身的消息丢失，响应延迟时间和带宽网络输出进行比较。 结果表明，本研究中提出的框架导致消息丢失减少。

关键词：WebSocket、 HTML5、 发布/订阅、消息代理

1. 介绍

在网络技术的成长过程中，网络应用的实时数据的使用将在未来几年增加[1]。随后，对实时网站的需求将增加； 这包括在数据显示，统计，仪表板监视，通知，即时消息和多玩家游戏中的实时 web 应用方法的简单用例的特征。 基于将信息/消息推送到客户端而不是同时向服务器请求更新的实时网站可能导致非常不同的技术架构。 构建实时 Web 应用程序需要提供具有最小响应延迟时间的双向通信的连接，这带来了更响应式的用户交互。 应用程序还必须具有能够处理与客户端的消息交换的服务器。

1.1. WebSocket

本研究的构建实时 web 应用的方法是利用 WebSocket，这是一种通过单个 TCP 连接提供全双工通信信道的协议。WebSocket 协议通过提供一种标准化的方式，使服务器向浏览器发送内容，而不需要客户端请求作为触发器。 消息还可以在浏览器和服务器之间双向传递，同时保持打开的连接。 此外，WebSocket 技术的一些早期采用者已经提出了对 WebSocket 与企业中间件基础设施（EWMI）的兼容性的担忧，例如代理服务器，负载均衡器，防火墙，消息代理等[1]。
相对于 HTTP 协议，WebSocket 协议提供了更短的延迟时间[2]。 通过 WebSocket 连接进行数据传输的过程难以避免数据，特别是随着连接数量的增加。 当 WebSocket 会话失败时，协议不具有解析和定义消息那些已经接收和那些已经丢失的消息的能力[3]。 因此，分布式应用程序和更复杂的逻辑过程需要一个模型，该模型可以基于订阅时指定的约束来管理向客户端通知/消息分发。

1.2. 发布/订阅

发布/订阅模式是为分布式系统类提供异步通信的范例。 Message Broker 是一种基于用于消息验证，消息转换和消息路由的发布/订阅模式来调解应用程序之间的通信的体系结构。 发布/订阅系统使用户能够表达他们对订阅的兴趣，并最终基于他们对任何发布商生成的事件的兴趣接收到相应的通知[4]。

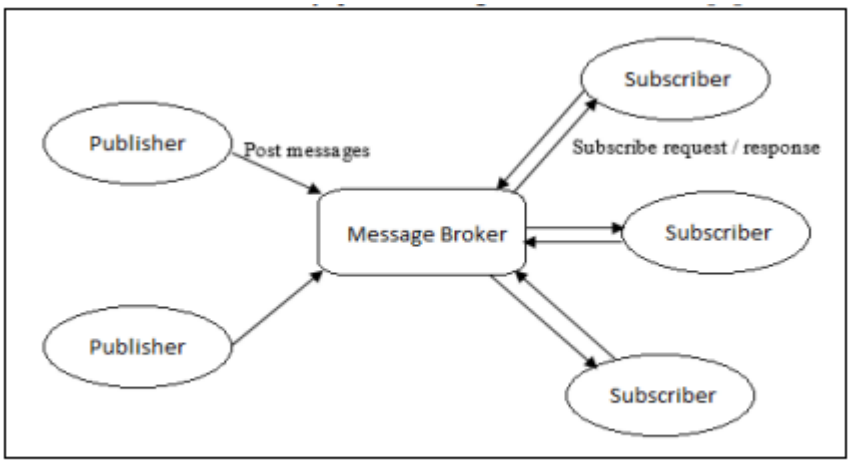

Fig.1 Publish/Subscribe Schema [5]

如图 1 所示，发布者向消息代理发出或发布消息。 订阅者必须在特定消息代理注册，这个代理将来会执行把发布者的相关消息转发到订阅者的任务。

1.3. 问题制定

随着 WebSocket 连接数量的增加，数据丢失成为搅乱的问题。然而，没有协议在服务器和客户端之间的数据传输过程期间处理和定义数据丢失问题。

1.4. 假设

假设如下：

1) H1：应用发布/订阅模式方法减少在 WebSocket 连接上的消息传输期间的消息丢失；
2) H2：应用发布/订阅模式方法会增加 WebSocket 连接的响应延迟时间；
3) H3：应用发布/订阅模式增加了 WebSocket 连接之外的带宽网络。

2. 相关工作

以前的研究提出了与 WebSocket 连接相关的模型解决方案，并将它们与用于实时数据通信的各种方法进行比较。 通过比较分析，发现 WebSocket 是一个提供 Web 服务器和客户端之间的有效通信，具有相对于 HTTP 轮询，AJAX 长轮询，COMET，HTTP 流和 WebSocket 的最低响应延迟时间[2，6] 的协议。 Pimentel 和 Nickerson [7]使用各种方法比较客户端和服务器之间的单向延迟，例如使用 WindComm 应用程序的 WebSocket 轮询和长轮询。

Alamsi 和 Kuma 评估了 WebSocket 的性能，实验表明 WebSocket 不仅带来更少的网络数据传输，而且通过抽象化数据的发送和接收过程显着简化了开发。此外，发布/订阅架构提供更好的可扩展性和可靠性，以避免消息拥塞，冲突和丢失[9]。

Khanaferov 确定了开发异步 Web 应用程序的最佳方法[10]。 本研究提出了使用 Java 消息服务作为消息代理的异步发布/订阅框架解决方案。 类似的方法也被 Roussele [11]使用。 Khanaferov 为异步服务器推送传输层实现了 Java Applet 插件[10]。 生成的框架由一个发布/订阅模式组成，以便于实现异步消息传递接口。 然而，由于增加的并发用户请求的造成的数据丢失问题仍然未解决，导致本研究的假设和问题制定。

3. 方法

这项研究本质上是定量的，旨在确定 Web 应用程序架构的性能和有效性，以提供建立实时的基于 Web 的应用程序的初步解决方案。 测量值包括拟建的具有发布/订阅模型的 WebSocket 解决方案与 WebSocket 连接（不具有发布/订阅模型）之间的响应等待时间的差异。 所得到的模型可以用作 web 开发者在设计实时的基于 web 的应用程序时的指南。

3.1. 研究设计

为了进行实验，构建了两个用于传输消息的 web 应用服务器。第一个实验应用程序服务器构建在 WebSocket 上，而不实现发布/订阅模式方法，而第二个实验应用程序服务器使用发布/订阅模式方法实现。用于这两个应用程序的技术是相同的，因为 WebSocket 技术是 HTML5 的一个特征。本研究使用 HTML5 来作为支持 WebSocket 连接的网站的内容结构和表示。 Javascript 代码用于客户端编程和连接建立。服务器端编程使用 Microsoft ASP.NET，它是一个 Microsoft 支持的技术，专为 web 开发生成动态网页，可以轻松开发和部署。 Microsoft ASP.NET 还拥有许多减少编码工作的功能，工具和类。 AWS SQS（Amazon Web Service 简单队列服务）是实现发布/订阅模式的消息代理。

两个实验情景都进行 30 次测试迭代。 每组迭代分为以下几种：

1) 来自 50 个并发用户的 50 条消息；
2) 来自 100 个并发用户的 100 条消息；
3) 来自 500 个并发用户的 500 条消息；
4) 来自 1，000 个并发用户的 1，000 条消息；
5) 来自 5，000 个并发用户的 5，000 条消息；
6) 来自 10，000 个并发用户的 10，000 条消息；

要模拟最多 10，000 个用户请求，使用了第三方工具 Loader.io。 Loader.io 是一个开源的，基于 Web 的应用程序，可以用来在客户端对应用程序进行压力测试。

两个实验情景的输出结果是响应延迟时间，与响应一起发送的消息和带宽网络输出。之后对输出数据进行处理和统计分析。

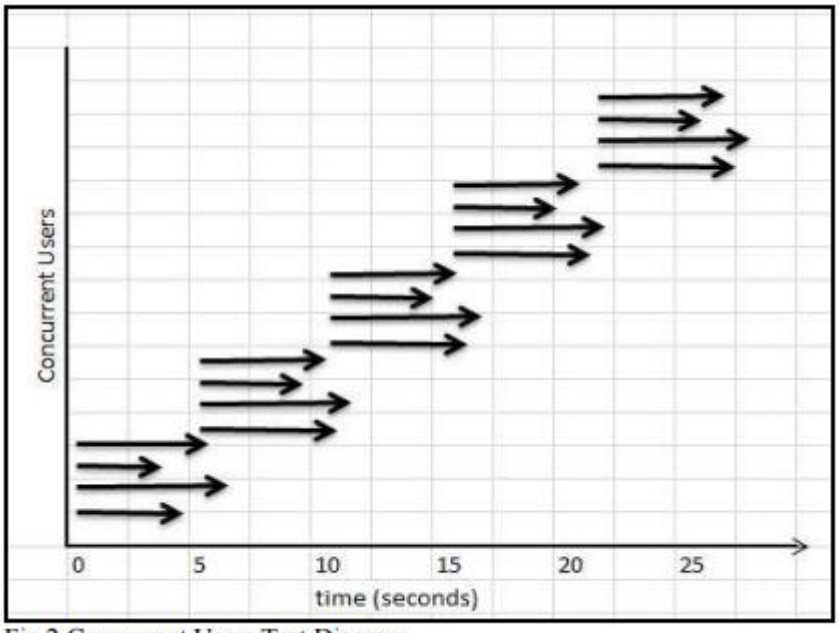

Fig.2 Concurrent Users Test Diagram

图 2 显示了测试的方法。 每个箭头表示向 Web 服务器发出请求的客户端。箭头的长度表示客户端在给定测试中访问服务器的响应时间。

3.2. 测试环境

测试环境在云计算平台 Amazon Elastic Compute Cloud（AWS EC2）上运行。服务器主机操作系统分配了专用的 CPU 内核，并配置了以下规格：

1) 操作系统：Windows Server 2012 R2 标准 64 位
2) 处理器：Intel®Xeon®CPU E5-2670 v2 @ 2.50GHz
3) 内存：1024 MB RAM
4) 应用服务器环境：IIS8、.Net Framework 4.5
5) 测试工具：Internet Explorer 11、Loader.io、SolarWinds 实时带宽监视器

3.3. 数据集

为了收集用于分析的数据，通过在 Web 应用服务器中建立环境来进行模拟，一个是具有发布/订阅模式的 WebSocket 的应用，另一个是没有发布/订阅模式的应用。 对于这两种环境，通过从客户端发送消息执行测试。 测试的输出结果是用于分析的数据。

3.4. 分析方法

该研究使用独立 t 检验统计分析。 T 检验分析比较两个实验组之间的响应等待时间。 使用相同的测试来查看受 WebSocket 连接上的应用发布/订阅模式的影响的带宽网络输出。

4. 结果

4.1. 数据丢失

TABLE I
DATA LOSS COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Pub/Sub)		Group 2 (with Pub/Sub)	
	Success	Loss	Success	Loss
50	100%	0%	100%	0%
100	100%	0%	100%	0%
500	99.24%	0.74%	100%	0%
1,000	98.86%	1.14%	100%	0%
5,000	94.30%	5.70%	99.02%	0.98%
10,000	91.71%	8.29%	99.15%	0.85%

通过比较发送到服务器的消息的总响应计数和成功响应计数来分析数据丢失。

表 1 显示了对于不同的并发用户，通过 30 次迭代发送的每个响应的响应成功的结果。成功率由成功响应总数除以发送的总请求数计算得出。数据丢失率通过从 100％减去成功率百分比得出。

基于表 1，可以看出，在组 1（没有发布/订阅）的前两种情况下，不存在数据丢失。对于具有 500 个并发连接的第三种情况，发生 0.74％的数据丢失率。此外，在较高负载连接（1，000，5，000 和 10，000 并发连接）下，数据丢失率同时增加。对于组 2（具有发布/订阅），一旦并发用户连接达到 5000，数据丢失发生，尽管成功率高于组 1 的成功率。对于 10，000 连接的情况，组 2 数据丢失相对组 1 减少超过 7％。尽管有这种改进，第 2 组仍然出现了数据丢失。但是，丢失的消息可以重新排队到 AWS SQS 消息代理，将处理数据丢失问题的任务委派给消息代理来解决。

TABLE II
DATA LOSS T-TEST RESULT BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value ($\alpha = 0.05$)
	N	Mean (Total Data Lost)	Std. Deviation	N	Mean (Total Data Lost)	Std. Deviation	
5,000	30	285	2.45	30	49	1.73	< 0.05*
10,000	30	829.20	18.62	30	85.20	2.23	< 0.05*

*significant at $p < 0.05$

因为应用发布/订阅模式的组 2 对于 50，100，500 和 1000 个并发用户没有消息丢失，所以使用独立 t 检验方法来比较两个组的 5，000 个并发用户连接和 10，000 个并发用户连接， 如表 2 所示。 对于 5，000 个并发用户和 10，000 个并发用户，p 值小于 0.05 的 α 水平，接受假设 H1。 换句话说，将发布/订阅模式应用于 WebSocket 可显着减少 WebSocket 连接上的消息传输过程中的数据丢失。

4.2. 响应等待时间

通过测量发送消息和客户端接收到消息时的时间戳之间的差异来计算响应等待分析。

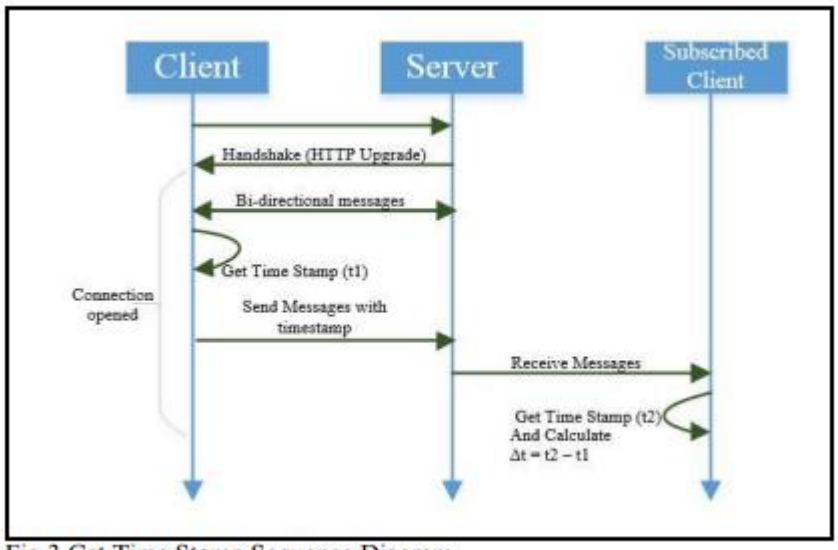

Fig.3 Get Time Stamp Sequence Diagram

图 3 显示了“获取时间戳”的序列，用于计算响应等待时间。 在向服务器发送消息之前检索时间戳，然后当订阅客户端接收到消息时再次检索。

TABLE III
DATA LOSS COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value (α = 0.05)
	N	Mean (ms)	Std. Deviation	N	Mean (ms)	Std. Deviation	
50	30	36.83	2.89	30	38	1.70	0.06168
100	30	87.67	3.97	30	89.37	2.68	0.05681
500	30	292.77	6.04	30	300.20	3.4	$2.21x10^{-7}$*
1,000	30	489.43	7.76	30	497.50	4.53	$7.56x10^{-6}$*
5,000	30	678.87	11.27	30	688.97	4.87	$3.25x10^{-5}$*
10,000	30	793.63	10.50	30	805.30	5.48	$1.32x10^{-6}$*

*significant at $p < 0.05$

表 3 示出了对于 50，100，500，1，000，5，000 和 10，000 个并发用户的每个测试用例的组 1 和组 2 的响应等待时间 t 检验的结果，其中每个测试用例进行 30 次迭代测试。

对于 50 个并发用户情况，具有发布/订阅模式的组 2 具有较高的响应等待时间平均值，具有大约 1.17 毫秒的差异。 然而，组 2 的标准差低于组 1 的标准差，表明前者在 30 次迭代中具有更稳定的响应等待时间。 0.06168 的 p 值高于 0.05 α 水平。 因此，对于 50 个并发用户连接，两个组之间的响应等待时间没有显着差异。

对于 100 个并发用户的情况，组 2 还是具有更高的平均响应等待时间，但是具有比组 1 更低的标准偏差。0.5681 的 p 值大于 0.05 α 水平，说明 组 1 和组 2 在 100 个并发用户连接时的响应等待时间没有显着差异。对于 500，1，000，5，000 和 10，000 个并发用户情况中的每一个，组 2 具有相同的的模式，即具有较高的响应等待时间平均值但低于组 1 的标准偏差。每种情况的 p 值低于 0.05 α 水平 。 因此，对于 500，1，000，5，000 和 10，000 个并发用户连接，响应延迟时间存在显着差异。

4.3. 带宽网络输出

TABLE IV
BANDWIDTH NETWORK OUT COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value (α = 0.05)
	N	Mean (Kbps)	Std. Deviation	N	Mean (Kbps)	Std. Deviation	
50	30	10.52	0.23	30	10.51	0.21	0.9392
100	30	18.92	0.61	30	19.03	0.51	0.4809
500	30	44.47	1.16	30	44.80	1.23	0.2944
1,000	30	70.24	0.77	30	70.80	0.77	0.0074*
5,000	30	286.58	2.21	30	298.44	3.11	$3.73x10^{-24}$*
10,000	30	307.25	4.61	30	332.44	4.94	$3.54x10^{-28}$*

*significant at $p < 0.05$

带宽分析通过带宽网络的平均值来进行，从客户端向服务器发送请求开始，直到订阅的客户端从服务器得到响应。

独立的 t 检验分析也用于查看带宽网络输出如何受 WebSocket 连接上的发布/订阅模式的应用的影响。

表 4 显示了组 1 和组 2 的带宽消耗结果，每个组使用 50，100，500，1，000，5，000 和 10，000 个并发用户请求进行测试，每种情况下使用 30 次迭代。 带宽网络输出使用量随着更高的并发用户连接而增加。 组 1 和组 2 之间的带宽间

隙也随着并发用户连接而增加。 对于 50 个并发用户连接，组 1 和组 2 之间的平均带宽差距为 0.01 Kbps。对于具有 10，000 个并发用户连接，间隙是 25.19Kbps。对于 50，100 和 500 个并发用户连接，两组之间的平均值和标准差的差异很微小。此外，每个 p 值高于 0.05 α 水平。因此，在这些情况下，组 1 和组 2 之间的带宽网络输出消耗没有显着差异。相比之下，对于 1，000，5，000 和 10，000 个并发用户连接，每个 p 值低于 0.05 α 水平。这表明在这些情况下，组 1 和组 2 之间的带宽网络输出消耗有显着差异。

从 t-test 结果可以得出结论：

1) 50，100 和 500 并发用户连接时，应用发布/订阅模式的 WebSocket 连接没有显着增加带宽网络输出。因此对于这些测试用例，假设 H3 被驳回。
2) 应用发布/订阅模式导致在 1，000，5，000 和 10，000 并发用户连接的 WebSocket 连接中带宽网络显着增加。因此对于这些测试用例，假设 H3 被接受。

5．结论

在 WebSocket 连接上实现了发布/订阅模式模型，将它和 WebSocket 本身就消息丢失，响应等待时间和带宽网络占用三方面进行比较。 测试有两组：第一组是 WebSocket 连接本身，第二组是应用了发布/订阅模式的 WebSocket。 每组对不同数量的客户端连接执行 30 次迭代。 结果表明，具有发布/订阅模式的组具有可以降低超过 7％的消息丢失，但是以更高带宽网络输出消耗和更高响应等待时间为代价，特别是在客户端连接数量较高时（在 1000- 10，000 并发用户）。

[参考文献]

[1] Alamsi, A., & Kuma, Y. Evaluation of WebSocket Communication in Enterprise Architecture. Available: http://presentation.amiralmasi.com/ Evaluation of WebSocket Communication in Enterprise Architecture_ AmirAlmasi_YohanesKuma.pdf, 2013.

[2] Liu, Q., & Sun, X. Research of Web Real-Time Communication Based on Web Socket. Int. J. Communication, Network and System Sciences, Vol. 5, No 12, 2012, pp. 797-801.

[3] Hapner, E. M., & Suconic, C. The MessageBrokerWebSocket Subprotocol. Available: http://tools.ietf.org/search/draft-hapner-hybimessagebroker-subprotocol-01, 2011.

[4] Li, W. A Content-based Publish/Subscribe Framework over Structured Peer-to-Peer Networks, Master thesis, computer science, University of British Columbia, Vancouver, Canada. 2008. Available: http://www.cs.ubc.ca/grads/resources/thesis/Nov08/Li_Wei.pdf.

[5] Hauer, J. H., Handziski, V., Köpke, A., Willig, A., & Wolisz , A. A Component Framework for Content-based Publish/Subscribe in Sensor Networks, in Proc. of 5th European Conference on Wireless Sensor Networks (EWSN 2008), Bologna, Italy. 2008.

[6] Rakhunde, S. M. Real Time Data Communication over Full Duplex Network Using Websocket.IOSR Journal of Computer Science, Vol. 5, 2014, pp. 15-19.

[7] Pimentel, V., & Nickerson, B. G. Communicating and Displaying Realtime Data with WebSocket. IEEE Internet Computing, Vol. 16, No. 4, 2012, pp. 45-53.

[8] Mühl, G. Large-Scale Content-Based Publish/Subscribe System. 2002. Available under Simple publication rights for ULB.

[9] Yan, Z. L., & Dai, M. A. Realtime Group Communication Architecture Based on WebSocket. International Journal of Computer and Communication Engineering, Vol. 1, No. 4, 2012, pp. 408-411.

[10] Khanaferov, D. Publish-Subscribe Architecture for Building NonPolling Asynchronous Web Applications, Master thesis, computer science, California State University, Northridge, US. 2013. Available: http://scholarworks.csun.edu/handle/10211.2/3233. [11]

Rousselle, P. Implementing the JMS Publish/Subscribe API. Dr. Dobb's Journal, Vol. 27, No. 4, 2002, pp. 28.