РАЗПРЕДЕЛЯНЕ НА НАТОВАРВАНЕТО МЕЖДУ СЪРВЪР И КЛИЕНТ В JAVASCRIPT ПЛАТФОРМА ЗА ОБРАБОТКА НА ИЗОБРАЖЕНИЯ

Емил Ангелов

Пловдивски университет „Паисий Хилендарски“, Факултет по математика и информатика, гр. Пловдив бул. "България" 236

e-mail: emil.angelov@gmail.com

ВЪВЕДЕНИЕ

Разпределянето на задачите между клиент и сървър, според определени критерии е проблем, който има много аспекти. Аналогията може да бъде направена със задачата за валидиране на данни – валидирането може да се извърши от страната на клиента или на сървъра, или и на двете места, а може и да се разпредели. В общия случай, обработката на изображения изисква много ресурси. Работа само при клиента е несигурна, защото зависи от характеристиките на клиентската машина. От друга страна, прехвърлянето към сървъра и обратно изисква трафик и скорост в двете посоки. В статията се търси решение на този проблем, като се изследват комбинации от променливи фактори като: конкретната обработка, условията при клиента, параметрите на изображението, моментната скорост на връзката, възможностите на сървъра /в частност локален/... Вариант е обработка на миниатюра при клиента, докато оригинала се качва на сървъра и се обработва там. Съществува и въпрос: къде да се съхраняват междинните и финалните резултати от обработката. Показването им на клиента под формата на умалени копия – работни миниатюри, повишава ефективността като цяло. Трафикът е по RTC канал или по FTP, след тест и сравнение. Използва се AJAX - концепцията за асинхронна комуникация, като се цели минимизиране на трафика за максимум обработки за единица време. При мощен сървър и добра връзка, с едно „качване“ и „сваляне“, и помежду им – паралелната обработка на миниатюра при клиента и оригинал на сървъра е идеалният случай. По-вероятно обаче е, миниатюрите при клиента – браузър, да играят контролно - представителна роля. Крайните варианти са повод на изследване на вариативно-междинен процес на разпределяне на задачите. Междинният трафик - свит до дефинирани скриптови обозначения на обработките с техните параметри. В статията са представени резултати и изводи, базирани на изследване чрез промяна на различни параметри.

ПОСТАНОВКА НА ЕКСПЕРИМЕНТА. РЕЗУЛТАТИ.

В **постановката на задачата,** параметрите, които се изследват, са:

- Обработки;
- Клиентска машина;
- Сървър;
- Връзка.

Популярни Автоматизирани Поточни **Обработки** (PASP) за целта на теста: (без намеса на оператор)

- Общи корекции на тон, цвят, острота (AutoTone, AutoColor, SmartSharpening);
- Преоразмеряване / Resize към два размера, в посока минимизиране – DownSampling;

- Записване (съхраняване) в различни размери и файлови формати – JPG, PNG, TIF.

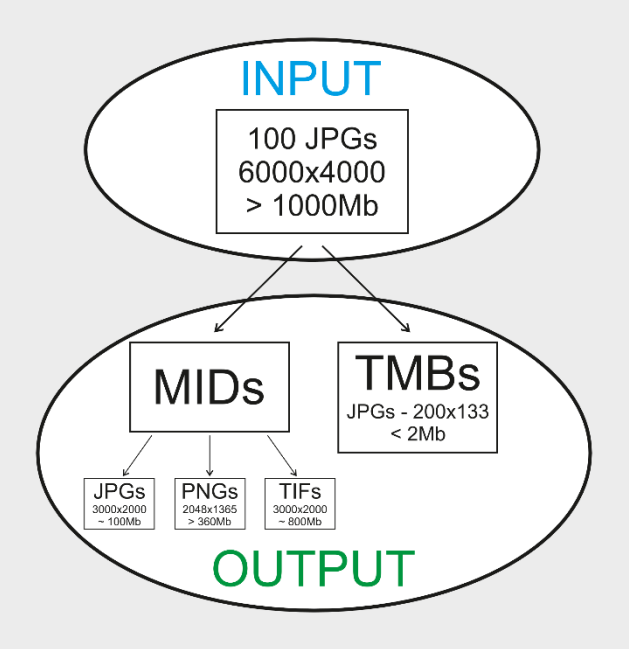
За изследването са използвани 100 броя изображения в папка, със следните характеристики - 24 мегапикселови снимки, JPG формат, 6000x4000 пиксела, от различни фотографски жанрове – портрети, пейзажи, репортажи, хаотично разпределени, със среден обем от по над 10 мегабайта всяка (около 1Гб общ обем). Модулите, извършващи всички последователни елементарни операции са написани на node.js v14.xx, като части от JS среда за обработка на изображения.

Пакетната поточна обработка използва на входа гореописаната папка, като на изхода генерира 5 папки, както е показано на фиг. 1:

В папката MIDs подпапките са три, както следва:

- JPGs – форсирано конвертирани към sRGB профил с компресия 10/12 (със загуби), на размер 3000x2000, с общ обем около 100Mb;
- PNGs – за Фейсбук – също с конверсия към sRGB, оптимизирани (компресия без загуби), на размер 2048x1365, с общ обем над 360Mb;
- TIFs – за печат – конверсия към AdobeRGB профил, с LZW компресия без загуби, на размер 3000x2000, с общ обем около 800Mb.

В папката TMBs миниатюрите са получени от JPGs, с компресия 5/12 (със загуби), на размер 200x133, с обем под 2Mb общо.



Фиг. 1. Схема на входни и изходни данни

При определяне на фактора **клиентска машина** са проучени множество, от които са избрани три, с различни количества и скорости RAM и различни по мощност процесори. За статията ги определяме качествено като – силна, средна и слаба. **Слабата** клиентска машина е лаптоп HP ProBook 450 G3 Base Model NoteBook PC със Intel Core i5-6200U 2.30 GHz (3 MB), 8 GB DDR4 SO-DIMM, 128 GB SATA SSD. **Средната** машина е отново лаптоп Lenovo Workstation W541 с 16 GB DDR3, Intel Core i7-4810MQ 2.80 GHz, 256 GB SATA SSD. **Силната** клиентска машина е настолен компютър - HP Z2 G5, Intel Core i7-10700 (2.9/4.8 GHz, 16M), 32 GB DDR4 3200 MHz, 512 GB SSD M.2 NVMe.

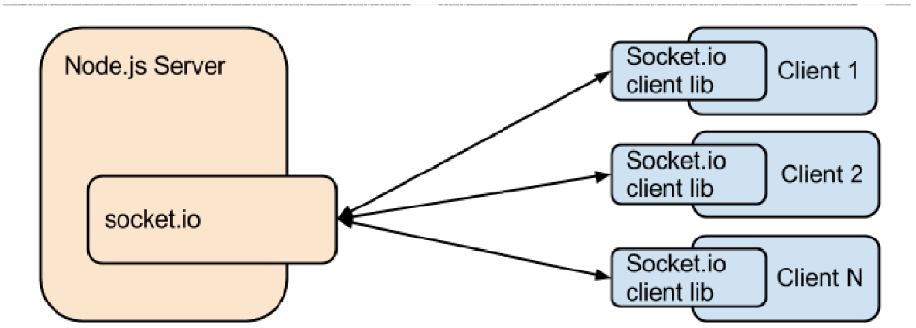
Изборът на клиентски машини за провежданото изследване беше продиктуван от стремежа факторът скорост на обработките да бъде чувствително различен. В случая с избраните клиентски машини имаме приблизително следните времена за изпълнение на гореописаната обработка:

- Слаба машина – около 8 минути;
- Средна машина – около 3.5 минути;
- Силна машина – около 1.5 минути.

Трите клиентски машини са с ОС MS Windows 10. Тествани са обработките и на Linux и на MAC OS. При аналогични мощности на хардуера, са получени приблизително същите времена за обработка. Това може да покаже, че и трите OS са оптимизирани достатъчно, за да не дават значима разлика при съизмерими хардуерни мощности.

Изборът на параметър **Сървър** бе силно затруднен, след избора на силната клиентска машина. Една от поставените експериментални цели бе сървърът да извършва най-бързо обработката и ако е възможно – не в проценти, а в пъти по-бързо, сравнено с клиентските машини. Поради тази причина, са използвани услугите на доставчик на хостинг услуги, който предостави за целта на експеримента Managed VPS хостинг план, който включва машина с Линукс базирана ОС – FreeBSD, на която инсталирахме модулите за обработка. Изтествани са няколко различни хостинг плана (Duo, Pro, Mega, Business), всеки от които с различни параметри, но при всички тях времето за обработките беше над 1 минута. Последващия експеримент беше проведен с план Business++, и при него беше постигнато време от 30 секунди. След известни оптимизации на настройките на сървъра, времето за обработка беше минимизирано до 24 секунди – стабилно време за всички опити.

При клиентските машини има известен фактор – наработка, като най-добрите времена се получават поне след 3 поредни опита. Тук следва да отчетем факта, че операционната система MS Windows работи най-добре с последно стартираните процеси.



Фиг. 2. Схема на Socket.IO

Факторът **Връзка** е интересен с това, че е вариативен и в немалко случаи е неподвластен за управление. Да уточним, че връзката между клиента и сървъра е двупосочна. Да приемем, че необработените изображения се намират при клиента, както и че при него желаем да се доставят резултатите от обработката. Ако нещата се извършват последователно, т.е. файловете първо се качват на сървъра, след това се обработва на сървъра и накрая резултата се връща обратно при клиента, то получаваме резултат, сравним по време или дори по-слаб от това, обработката да се извърши директно на средната или бързата клиентска машина, без участие на сървъра. Оправдание за това може да имаме само в случай, че клиентът не желае или не може да има на машината си обработващите модули. Оптимизацията, в която роля играят параметър **Връзка,** както и **поточността** на обработките, може да бъде осъществена чрез избор измежду различни канали. За целта е разработен модул, който именувах **TTS** (Traffic Test and Switch) за най-добър трафичен резултат. От една страна, тестваме един стабилен **FTP** канал, а от друга - използваме node.js server апликация със сървърно socket.io, като част от интегрираната JS среда, в едно с клиентско socket.io. Предимството тук е, че сървърното socket.io може да обслужи много

клиенти паралелно или един клиент с голяма оптимизация на връзката при слаби физически параметри (фиг. 2).

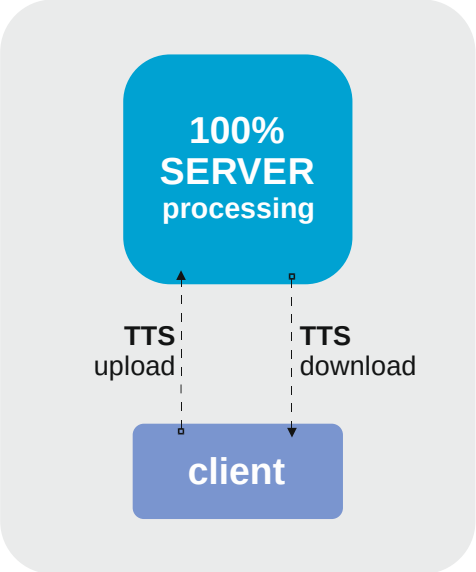
Интересното е, че FTP скоростта пада при много малообемни файлове и се закрепя стабилно при малък брой големи файлове. TTS следи и отчита в реално време тези промени. Отлични резултати се получават, когато големите файлове минават през FTP, а множеството малки файлчета – миниатюрите през сокета.

Има още един резервен канал, включен в приложението – WebRTC модул. Той предимствено е за връзка между различни клиенти (p2p), но в частният случай, при него едната страна може да е нашият сървър. Добрата му характеристика е, че във всеки момент прекарва трафика през най-бързите пътища, нещо, което не е така при сокетите и FTP. Той е най-стабилен при нестабилна връзка и най-сигурен за безаварийното доставяне на файловете в двете посоки. Дори при прекъсване на връзката и след възстановяването ѝ не се налагат повторни опити. При трите вида връзки се използва Хофманов контрол за интегритета (цялостност и здравина до байт) на файловете. FTP вграденият контрол води до повторение на качванията на цели файлове, което не е приемливо при големите файлове. Сокетите контролират на пакети и там повторенията им водят до по-малки загуби на време. При слаба, нестабилна или много променлива връзка, WebRTC спасява положението. В случай, че имаме сигурна връзка, може да го изключим от проверките. Това също е в обхвата на TTS. Тук ще обобщя, че благодарение на TTS, при слаба връзка имаме средно приблизително удвояване на скоростта или двукратно съкращаване на времето за пренос на снимките, спрямо най-бързия от трите канала, използван самостоятелно. При нашите тестови данни – **качване** над 1000 Mb и **сваляне** над 1260 Mb разделям връзките на 3 вида:

- Слаб трафик > 30 минути;
- Среден трафик ~ 15 минути;
- Силен трафик < 5 минути.

Тук тестът измерва само времетраенето на трафика, като не се отчита времето за обработка на сървъра. Изображенията са предварително обработени и готови за сваляне. TTS повишава ефективността и чрез разделяне на каналите за качване и сваляне, и паралелното им изпълнение.

Цялостните реални тестове работят, като отчитат степента на наслагване на времената на трафика и обработката. Трябва да се отчита и системата за мониторинг при клиента – дали е включена или изключена – обработките се правят „на тъмно“.



Фиг. 3. Начален експеримент – цялата обработка е изнесена на сървъра

При един от началните експерименти е тестван крайния случай – 100% обработката е изнесена на сървъра – без мониторинг (фиг. 3). При наличие на силна машина и силен трафик, общото времетраене на процеса беше **5:05 /пет минути и пет секунди/.** Става очевидно, че трафикът заема най-големия времеви дял в експеримента.

При така подбраните параметри, силната и дори средната машина се справят самостоятелно със задачата за по-кратко време.

Следващите тестове са насочени в друга посока – опити за подобряване на това време и **добавен мониторинг** на сървърните обработки при клиента. За уточнение – фиксираме мониторинга до процес, при който сваляме генерираните миниатюри от папка TMBs при клиента, паралелно на обработките и генерирането на изображенията за съответните папки на сървъра с подходящи съобщения като: DSC_1112.jpg (3000x2000) в JPGs, така също и за PNGs и TIFs. Резултатът е абсолютно същият **5:05 /пет минути и пет секунди/.** Добавката на мониторинг с пренебрежимо малък трафик, в сравнение с общия, който се показва на клиентската машина и в случая спестява свалянето на папка TMBs, накрая явно не натоварва видимо сървъра, който спаралелява обработките. Една нишка в повече, и то съвсем тъничка... При избор на Сървърен план има коефициенти x2, x3 и т.н. добавена процесорна мощност, като паметта динамично се удвоява, в зависимост от работата на приложението.

Казано с други думи – сървър, без никакви хардуерни лимитации. За съжаление високата цена на тази облачна услуга ограничава потребителите, които могат да я ползват.

Тук следва да отбележим, че фотообработките са само един конкретен вид обработки, които реално може да бъдат подменени с други такива – като например намиране на криптоключове, задачи от астрономия, физика, медицинска генетика и други области. Т.е. представеното изследване може да се разшири в посока – разпределяне на обработки между клиент и сървър, в нашите тестове дотук – прехвърляне на обработки на мощен сървър, а съвсем абстрактната посока би могла да бъде – разпределяне на обработки в многомашинен клиентски слой.

Понеже става ясно, че не можем да се отървем от голямата тежест на преноса на данни (докато чакаме 5G технологията да навлезе масово), започваме серия експерименти без отчитане на времето на трафика за качване на входните данни. Приемаме, че изначално ги съхраняваме в облачното пространство, в случая – нашият Сървър. Ако приемем, че резултатите от обработката също остават там, в едно с увеличаване на скоростите на обмен се увеличават и обемите на достъпните облачни пространства, то тогава имаме разглеждане на участието на клиентската машина в подобряване на общата ефективност на процеса. Не е маловажно и участието на потребителя. Освен мониторинг, той заявява и желание за контрол над процеса на обработка.

Ако в началния пример има поточна пакетна обработка без пряка намеса на потребителя или той е само наблюдател, то в следващия експеримент потребителят вече има активно участие. Той избира изображения и определя реда за обработки. Това става чрез интерактивния модул на JS средата. На клиентската машина се сваля папката TMBs с изображения за контрол и мониторинг. Тук вече миниатюрите са малко по-големи (600x400), за по-добра видимост на операциите над тях. Обработките са от същия набор, като от горния пакет. Разликата в случая идва от намесата на потребителя при избор на изображения и ред на обработки. Това става чрез участие на малък команден пакет – списък допустими команди с параметри – участващите изображения. Реализиран е като команден интерпретатор, чрез който към сървъра се подават команди. Паралелно се извършва обработка и визуализация на миниатюрата при клиента, докато на сървъра се извършва съответната обработка над оригиналното изображение. Във фонов режим резултатите може да се свалят при клиента, ако той е пожелал, докато се извършват обработките на следващите изображения.

И така, след интеракцията от страна на клиента, без отчитане на времето за качване на входните изображения, след като клиентът е избрал всичките 100 снимки от папката INPUT и е указал същата последователност на действията, като при автоматизираният тест, и с отчитане на времето за сваляне на резултатите, постигаме резултатите, показани на табл. 1 и табл. 2. Времената са в минути и секунди. Ясно се забелязва отново тежестта на трафичното време. При силния трафик, клиентската машина вече има минимално значение като фактор.

Трафик \ Машина	Слаба	Средна	Силна
Слаб	10:10	10:00	09:50
Среден	05:05	05:00	04:50
Силен	01:20	01:20	01:20

Табл. 1. Резултати с отчитане на времето на трафика за качване на входните данни

Трафик \ Машина	Слаба	Средна	Силна
Слаб	00:50	00:48	00:44
Среден	00:38	00:35	00:33
Силен	00:30	00:30	00:30

Табл. 2. Резултати без отчитане на времето на трафика за качване на входните данни

Тук вече сме съвсем близо до сървърното време. Забелязва се, че сваляне е по-бързо от качването по това, че в този случай нямаме изчакване на обемите за сваляне.

ЗАКЛЮЧЕНИЕ

В заключение можем да направим следните обобщения – при мощен сървър, заглавието на статията от „Разпределяне на натоварването...“ би станало „Прехвърляне на натоварването към Сървъра...“, но засега, все още е възможно част от обработките да се осъществяват при клиента в „реално време“ – сходно или малко по-добро от клиент-сървърния вариант. Но тенденцията е ясна – с подобряване на факторите скорост и надеждност на връзката, както и увеличаването на обемите на достъпно облачно пространство, обработките постепенно ще се изместят в посока по-далеч от клиента – само във физическия смисъл. Иначе казано, клиентът с неговата машина си ще са само в терминална роля и много по-близко до облачното пространство.

Средата за разпределяне предстои да бъде развивана в двупосочна командна интерпретация – отдолу-нагоре – команди от клиента към сървъра, отгоре-надолу – известяване на клиента от сървъра с миниатюризирани обработки, съпроводени с текстови съобщения и друга индикативност за статуса на обработките.

Благодарности: Работата е подкрепена от проект МУ21-ФМИ-004, финансиран от Фонд „Научни изследвания“ при Пловдивския университет „П. Хилендарски“.

ЛИТЕРАТУРА

1. Хавербек, М., Eloquent JavaScript, 2018, Глава 19, Проект: програма за рисуване, [https://to6esko.github.io/19_paint.html, 20.08.2021].
2. Angular, [https://angular.io/, 20.08.2021].
3. Aurelia, [https://aurelia.io, 20.08.2021].
4. Backbone.js, [https://backbonejs.org, 20.08.2021].
5. Bootstrap, [https://getbootstrap.com/, 20.08.2021].
6. DoFactory, JavaScript Design Patterns, [https://www.dofactory.com/javascript/design-patterns, 20.08.2021].
7. DoFactory, JavaScript Introduction, [https://www.dofactory.com/javascript, 20.08.2021].
8. Ember.js, [https://emberjs.com/, 20.08.2021].
9. jQuery, [https://jquery.com/, 20.08.2021].
10. Haverbeke, M., Eloquent JavaScript, 2018, 3rd edition, [https://eloquentjavascript.net, 20.08.2021].
11. Haverbeke, M., Eloquent JavaScript, 2018, 3rd edition, Chapter 19 Project: A Pixel Art Editor, [https://eloquentjavascript.net/19_paint.html, 20.08.2021].
12. MDN Web Docs, [https://developer.mozilla.org/, 20.08.2021].
13. MDN Web Docs, Pixel manipulation with canvas, [https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas, 20.08.2021].
14. Meteor, [https://www.meteor.com/, 20.08.2021].
15. Microsoft Docs, [https://docs.microsoft.com/, 20.08.2021].
16. Mithril, [https://mithril.js.org, 20.08.2021].
17. npm, [https://www.npmjs.com/, 20.08.2021].
18. Polymer Project, [https://www.polymer-project.org, 20.08.2021].
19. React, [https://reactjs.org/, 20.08.2021].
20. W3schools, HTML Canvas Graphics, [https://www.w3schools.com/html/html5_canvas.asp, 20.08.2021].