Proof of Concept

Студент 1 - Јелена Хрњак PA86/2018

Студент 2 – Кристина Стојић PA96/2018

Дизајн шеме базе података

Дизајн шеме базе података креира у PowerDesigner-у се налази у фолдеру под називом ‘ClassDiagram’.

Предлог стратегије за партиционисање података

У нашој апликацији фокус је на резервацији више врста ентитета (викендица, бродова и авантура) и самим тим број података врло брзо може да достигне лимит јер се резервације никада не бришу већ се само могу отказати (што их не уклања из базе). Код великих софтвера са великим бројем података, приступа се партиционисању података. У нашем случају, адекватно би било приступити вертикалном и/или хоризонталном партиционисању података.

Хоризонталним партиционисањем, можемо поделити табеле са великим бројем објеката на више мањих. Табеле које би скоро засигурно имале највећи број података су табела са резервацијама и слободним терминима. Термине и резервације бисмо могли да поделимо хоризонталним партиционисањем по id-евима власника/инструктора. Проследили бисмо id власника или инструктора hash функцији која би одредила на ком серверу ће сместити или потражити резервацију/термин. Мана је што не можемо претпоставити да ће сви власници имати приближан број термина и резервација те би због оваквих осцилација у количини података неки сервери били вишеструко оптерећенији од других. Предлог који би решио овуману је партиционисање по типу ентитета, а затим партиционисање тих податке на основу власника/инструктора.

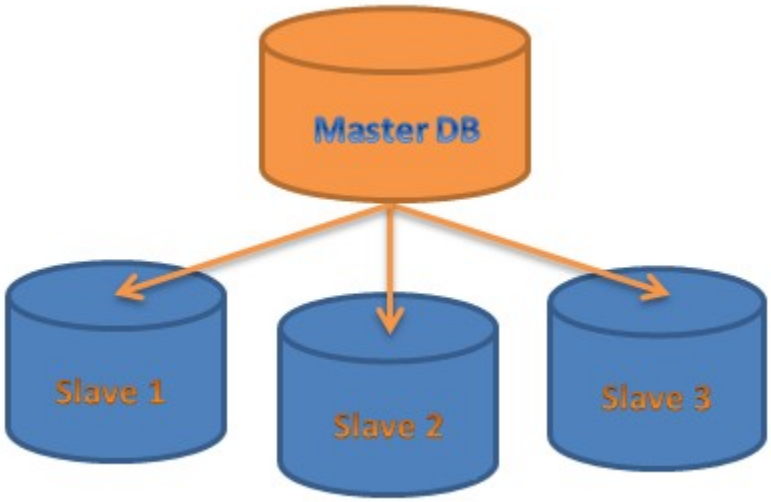
Још један предлог је да се резервације и термини партиционишу по времену почетка или краја. Резервације које су у даљој прошлости (поготово слободни термини) нису од великог значаја те бисмо их могли чувати независно од резервација и термина у

будућности или садашњости. Термине и резервације од мањег значаја бисмо могли чувати на серверима слабијих перформанси.

Поред овога, нешто што смо и ми применили при имлементацији је вертикално партиционисање табела са великим бројем података. Поделили смо иницијалну табелу User по улози и самим тим избегли смо пренатрпаност једне табеле и честе приступе истој ради измени или уписивању.

Предлог стратегије за репликацију базе и обезбеђивање отпорности на грешке

Предлог за репликацију базе би било увођење slave-master архитектуре. Секундарне (slave) базе би биле задужене за читање података (слободни термини, ентитети...) док би примарна (master) база обрађивала захтеве за писање. Такође, у master бази се чувају сви подаци, док се у slave базама чувају копије. Овим смо повећали перформансе јер су примарне базе растерећене од великог броја захтева за читање података и побољшано је очување података. Уколико би ипак дошло до преоптерећења master базе или уколико би из било ког разлога дошло до отказа, нека од slave база би постала master (док се проблем не уклони). С друге стране, ако би дошло до пада неке од slave база, master база би преузела захтеве базе која је онеспособљена. Оваквом архитектуром обезбеђена је отпорност на грешке јер су перфонмансе ојачане а и обезбеђено је очување података јер су они сачувани у више база.



Предлог стратегије за кеширање података

Ако претпоставимо да нашу апликацију користе молиони корисника, постојао би огроман број позива ка бази што би јако успорило читав систем. Како бисмо додатно убрзали процес комуникације клијента са сервером, односно смањили приступ нашим подацима добро би било извршити кеширање података.

Овај процес је најбоље извршити над статичким подацима који јсе ријетко мењају као што су слике, коментари и слично. Ако је ограничена кеш меморија најбоље је кеширати податке који се најчешће добављају као што су најповољнији/најпопуларнији ентитети, ентитети који садрже велики број акција као и периоди доступности истих. Овај процес можемо да имплементирамо уз помоћ Event Sourcing-а за праћење догађаја који су се десили у систему.

CND (Content Delivery Network) омогућава кеширање статичких фајлова на локацији која је географски најближа клијенту, односно на најближи proxy сервер који је повезан на оригинални сервер. На овај начин ће тражени подаци стићи ка кориснику много брже него да се сви захтјеви шаљу на главни сервер.

Оквирна процена за хардверске ресурсе потребне за складишење свих података у наредних 5 година

Претпоставке:

- укупан број корисника апликације је 100 милиона
- број резервација свих ентитета на месечном нивоу је милион
- систем мора бити скалабилан и високо доступан
- У просеку је резервисано 60% слободних термина
- У систему ће постојати по милион викендица, бродова и авантура годишње

Величина потребна за складиштење најчешћих ентитета:

- Address ~ 3.2 KB
- Usser ~ 6 KB
- Cottage ~ 8 KB
- Boat ~ 8 KB
- FishingAdventure ~ 10.5 KB
- Appointment ~ 2 KB
- Reservation ~ 3KB

Меморија потребна за претпоставке:

- 100 милиона корисника = 320GB
- милион резервација месечно је 60 милиона резервација = 180GB
- уколико постоји милион резервација, посотји око 40 милиона слободних термина = 80GB
- ентитети ~ 105 GB
- слободна процена за остале ентитете ~ 1500GB
  УКУПНО ЗА 5 ГОДИНА ~ 2 ТВ

Предлог стратегије за постављање load balansera

Уколико се не води рачуна о преоптерећењу сервера, врло брзо до њега може доћи. Load Balancer представља поделу оптерећења скалирањњем апликације и повећањем перформанси. Предлог је постојање „главног“ сервера који ће да распоређује захтеве на друге сервере у зависности од њиховог оптерећења. Једино о чему треба водити рачуна је да не сме постојати разлика у извршавању захтева, односно сваки сервер ком се проследи захтев треба да изврши захтев на исти начин.

Постоји више стратегија за постављање load balancera. Први предлог стратегије – **Round-robin,** један од најједноставнијих и најчешће коришћених алгоритама. Овај алгоритам би се могао користити у случају да су сви сервери истих или приближних карактеристика. Ово је алгоритам где се захтеви клијената дистрибуирају серверима у ротацији, те се на тај начин релативно равномерно распоређује оптерећење на све сервере. Уколико имамо два сервера, први захтев се прослеђује првом серверу, други другом, трећи првом серверу апликације и тако даље.

Међутим, уколико су сервери различитих перформанси, могло би се приступити алгоритму **Fixed Weighting.** Администратор сваком од сервера додељује тежину на основу разних критеријума (избор је на администратору). Захтеви се серверима распоређују на основу додељене тежине.

Предлог које операције корисника треба надгледати у циљу побољшања система

Како бисмо креирали добар софтвер, није само битно да он буде технички и визуално коректан, већ је један од најважнијих делова одговор корисника. Клијенти су ти који су извор профита власницима, инструкторима а и самој компанији која је креирала софтвер, те је од велике важности прилагодити им апликацију што је више могуће.

Једна од ствари која би се могла надгледати је у ком периоду дана или године се врши највећи број резервације. Да ли је то у току празника, у току лета итд. и самим тим прилагодити промоцију саме апликације и њених услуга тој информацији.

Поред овога, могли бисмо узети у обзир и који ентитети се највише заказују те их више препоручивати клијентима или увидети шта ти ентитети имају што немају они који се мање заказују (да ли је то фотографија, опис..).

Једна од највреднијих ствари које добијамо од клијената су оцене и жалбе, те бисмо на основу њих могли клијентима препоручивати добро оцењене ентитете. Истраживања су показала да ће клијенти пре заказати ентитет који је добро оцењен него неки без оцене или коментара. Могли бисмо у апликацији додати ,,популарне'' ентитете на основу оцена клијената и броја оцена, те би клијенти имали више поверења у викендицу или брод који заказују.

Комплетан цртеж дизајна предложене архитектуре