Portfolio

김도영
Doyoung Kim

Contents

1. Projects

2. Papers

3. Education

4. Awards

5. About Me

1. Projects

- Timeline
- Works / Projects
    - Recent Order

1 Timeline

2012
2013

Yonsei University
Undergraduate

Yonsei University
정보통신연구실

Time It

iOS 7
Swift 1.2

2014

2015

Yonsei University
Undergraduate

SkelterLabs

Gabe POS

GABE

Android
AngularJS
node.js
Appium

2016

2017

SkelterLabs

Hermes

React
Redux
Redux-saga
C++

Yonsei University
데이터공학연구실

Redis
NVRAM / PMDK
Vue.js
node.js
Jekyll

2018

2019

Yonsei University
데이터공학연구실

Redis
RocksDB
Docker
Nvidia GPU CUDA
Reactor (Java)

1 Works / Projects

- 연세대학교 데이터공학연구실 석사과정
  - 2018
    - Redis, NVRAM
    - Vue.js, node.js, Jekyll
  - 2019
    - Redis(Lettuce), RocksDB, Nvidia CUDA, Docker

1 Works / Projects

- Research
  - Redis
    - Redis-NVRAM Analysis
    - Persistent Buffer Redis
    - NVRAM-DRAM Hybrid Redis
  - RocksDB
    - RocksDB Analysis
    - RocksDB-GPU

- Projects
  - ADDB
  - GENAX
  - DELAB Homepage

Research

- Redis Analysis
  - AOF Logging
    - Redis의 Command를 Log file에다가 기록하는 기법
    - Background Thread에서 기록하므로, Persistence 정도에 따라 2가지 Policy로 나뉨
      - always
        - 1 fsync / 1 command
        - 장점: Full Persistence
        - 단점: Redis 성능 저하 O
      - everysec
        - 1 fsync / 1 sec
        - 장점: Redis 성능 저하 X
        - 단점: Weak Persistence

  - NVRAM (*Non-volatile Random Access Memory*)
    - 전원이 차단되어도 데이터가 유지되는 RAM
    - PMDK (*Persistent Memory Development Kit*)
      - Intel에서 개발
      - NVRAM에 Programming할 수 있는 C Library

- Redis Analysis
  - 세미나 자료
    - Redis_dicthash_sds.pdf

- **Persistent Buffer Redis**
  - Redis Project (Research)
    - 2018.04 ~ 2018.06
    - 언어 / 프레임워크
      - C, Redis

  - Redis의 AOF Logging(everysec)의 문제점을 NVRAM을 활용하여 개선

  - AOF Logging(everysec)의 문제점
    - Weak Persistence
    - Background Thread에서 쓰여진 Log가 Memory상의 Log Buffer에 잠시 저장되기 때문

  - AOF Log Buffer를 NVRAM에 구현
    - Full Persistence
    - AOF Logging(everysec)의 Performance 유지

  - Paper Link
    Github Link
    Persistent_Buffer_Redis.pdf

- NVRAM-DRAM Hybrid Redis
  - Redis Project (Research)
    - 2018.03 ~ 2018.10
    - 언어 / 프레임워크
      - C, Redis

  - Redis의 AOF Logging의 문제점을 NVRAM과 DRAM에 동시에 저장함으로써 개선

  - AOF Logging의 문제점
    - Weak Persistence
    - 단순한 Update임에도 Log가 계속 쓰여짐
      - AOF 파일이 굉장히 커짐
        - 필요 없는 Log를 삭제하기 위해 Rewrite Operation이 실행됨
        - Redis의 성능이 극심히 저하

- NVRAM-DRAM Hybrid Redis
  - NVRAM에 데이터를 먼저 저장하고, 상태에 따라 Cold Data를 DRAM에 Tiering
    - Full Persistence
    - AOF Rewrite 발생 빈도수 ↓
      - Why?
        - NVRAM에 먼저 저장함으로, Update가 일어나도 Logging이 발생하지 않음
        - Cold Data를 DRAM에 Tiering함으로써, Rewrite 발생률을 줄임

  - Paper Link
    Github Link
    NVRAM-DRAM_Hybrid_Redis.pdf

- RocksDB Analysis
  - Block-based SST File (SST = Sorted Static Table)
    - RocksDB는 LSM-Tree로 구조화된 SST 파일들에 데이터를 저장함
    - SST 파일의 포맷은 여러가지가 있으나, Block-based SST File을 기본 포맷으로 사용함

  - RocksDB Get Operation
    - Get(key)
    - 동작 과정
      - LSM-Tree Search로 key를 담고 있을 가능성이 있는 SST 파일들을 가져옴
      - 각 SST 파일 Search
        - Bloom Filter Checking (false positive)
        - SST File 구조에 따라 Search
    - 문제점
      - Single Key search
        - 굉장히 빠른 성능
      - Range Key search
        - 조회해야하는 SST 파일 개수가 증가
      - Single / Range Value search
        - 모든 SST 파일 조회 필요

- RocksDB Analysis
  - 세미나 자료
    - RocksDB-GPU.pdf
      - Introduction Part

- RocksDB-GPU
  - RocksDB Project (Research)
    - 2018.12 ~ 2019.06
    - 언어 / 프레임워크
      - C++11, Make, CUDA, RocksDB
  - 여러 SST 파일을 조회(Range Query, Value Search)를 GPU로 가속화
  - 구현:
    - Block-based SST File들을 다음과 같이 Collecting (CPU)
      - DataBlocks
      - DataBlock Offsets
    - Nvidia CUDA를 사용하여 Search / Filter 연산을 실행 (GPU)
      - 1 GPU Thread – 1 DataBlock Offset
      - CUDA Stream을 사용하여 Pipelining
  - 성능:
    - Query / Filter Selectivity가 낮을수록, Data가 많을수록 성능 향상
      - 최대 약 25% 향상
      - Copy from GPU to CPU가 Cost의 원인

Works / Projects

- RocksDB-GPU
  - Paper Link
    Github Link
    RocksDB-GPU.pdf

Project

Works / Projects

- ADDB (Analytic Distributed DBMS)
  - Spark / Redis / RocksDB Project
    - 2018.03 ~ 진행중
    - 언어 / 프레임워크
      - C, Redis/Lettuce/Jedis, RocksDB, Spark

  - 정보통신기술진흥센터(IITP) 주체의 정부과제 “SW스타랩” 프로젝트
    - IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발
    - Link

- ADDB (Analytic Distributed DBMS)
  - 본 과제는 다음 목표를 지향하고 있음
    - 개발 배경 / 원인
      - IoT 환경에서 파생되는 작은 크기의 막대한 양의 Big Data가 발생함
      - 이러한 Big Data에 대한 빠른 분석이 요구됨
      - 분산된 파일의 블록 단위로 수행하는 Hadoop 기반 시스템은, 작은 크기 대용량 데이터에 대한 대처가 어려움
    - 특징
      - 분산 환경에서 Big Data에 대한 빠른 분석 모듈 개발
        - Spark
      - DRAM – SSD 간의 데이터 배치 기술을 개발
        - Redis / RocksDB
      - SSD 하드웨어 및 NVRAM 구조에 소프트웨어를 최적화
        - RocksDB
  - Redis / RocksDB 데이터 배치 및 조회 모듈 개발에 참여함

- ADDB (Analytic Distributed DBMS)
  - Redis-RocksDB를 SparkSQL의 DB Engine으로써 구현
    - FpScan Command 구현
      - Redis / RocksDB에 저장된 Relational DBMS Data를 조회할 수 있는 API
      - 구현 Issue
        - RocksDB Get 시간 지연 문제
          - RocksDB의 Get Operation의 시간이 많이 소요됨
          - 여러 Column Data를 Vector로 묶어서(ColumnVector) 저장하여 해결
        - Redis Memory Explosion 문제
          - FpScan에서 너무 많은 Data를 Return하면서, Redis의 Memory가 극도로 팽창함
          - Metakeys command로 어느정도 해결하였으나, 문제가 여전히 발생
          - Lettuce의 Async 기능을 활용하여, Spark(Client) side에서 해결 시도 중
    - Metakeys Command 구현
      - Redis에 저장된 Relational DBMS Meta Data를 조회할 수 있는 API
        - Filter Predicate Push-down
      - Glob-pattern Search, Tree Search 지원
    - Redis2RocksDB Batch-Tiering 구현
      - Redis의 용량이 어느정도 차면, RocksDB로 Tiering
      - 여러 Row Data를 한꺼번에 Tiering하는 Logic
      - Evict Queue / Free Queue를 이용하여 Background Thread에서 동작

- ADDB (Analytic Distributed DBMS)
  - Lettuce Loader 구현
    - Redis의 Multi-thread 기반 Client인 **Lettuce**에 ADDB API를 구현
    - ADDB-Lettuce를 활용하여 Reactor / Future API 기반의 DataLoader를 구현
    - 구현 Issue
      - Thread Pool Hell
        - Reactor Scheduler로 elastic을 사용함
        - Scheduler를 Command Call마다 설정해서, thread가 굉장히 많이 사용됨
        - 과도하게 생성된 thread로 인해, 작업은 처리가 안되는데 GC가 굉장히 많이 호출됨
        - 해결
          - Scheduler를 설정하지 않고, native로 netty의 event loop에 맡김
          - Throughput 향상, GC 안정화
      - Future Memory Explosion
        - Command Call을 Future로 호출하고, Future List로 관리함
        - 마지막에 Future List를 한번에 await하게 구현
        - 하지만 시간이 지나면서 Memory 사용량이 급증하며 GC가 굉장히 많이 호출됨
        - 해결 (Progress)
          - Workaround로, 일정 사이즈의 Window로 Command 호출을 나눔
          - Window 단위로 Command를 Await

- ADDB (Analytic Distributed DBMS)
  - (De) Serialization by Google-Protobuf (Failed)
    - Redis의 Relational Data가 RocksDB로 Tiering될 때, Serialize 되어 저장됨
    - Serialization을 naïve하게 String으로 저장함
      ex) [1, 2, 3] → array:[1:2:3]

    - 문제점
      - Serialization의 구분자로 사용되는 문자가 들어간 데이터는 저장이 불가능함
      - (De) Serialization의 변환 비용, 저장 비용이 큼
    - 해결시도
      - Google-Protobuf로 Relational Data를 정의하여 사용하자!
        - (De) Serialization 비용 감소
        - 깔끔하게 코드 구성 가능
    - 문제점 재발생
      - Protobuf Message로 사용한 struct 구조체 사이즈가 너무 큼
        - Redis 용량 부하에 따른 성능 저하 > (De) Serialization 개선에 따른 성능 향상
      - Protobuf Message를 최대한 가볍게 만들었으나, 구조체 사이즈 개선이 되지 않음
      - 기존 naïve serialization을 계속 사용하고, P2로 넘김

- ADDB (Analytic Distributed DBMS)
  - Github Link (ADDB-Redis/RocksDB)
    Github Link (ADDB-Loader)
    Github Link (ADDB-Lettuce)
    Page Link

  - 세미나 자료
    - ADDB_fpscan.pdf
      ADDB_metakeys.pdf

Works / Projects

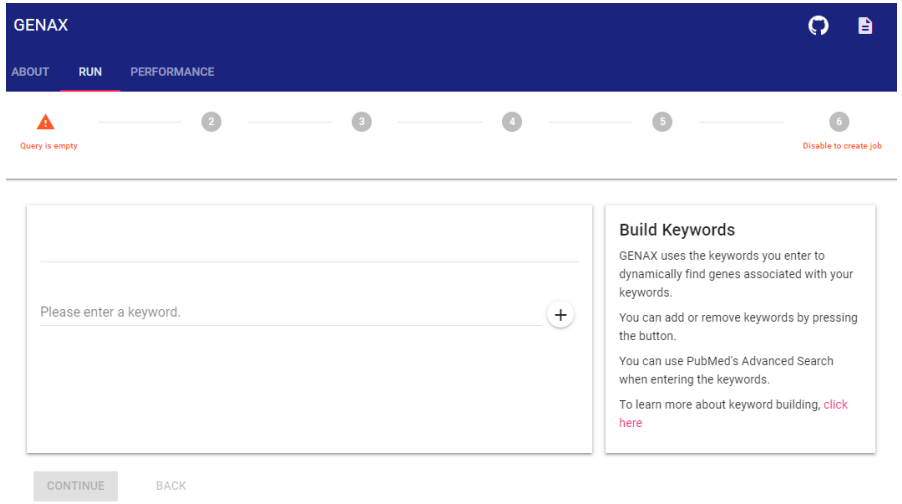
- GENAX
  - Web Application Project
    - 2017.03 ~ 2018.02
    - 언어 / 프레임워크
      - Vue.js, node.js, Python, Bash Script, MySQL, Docker

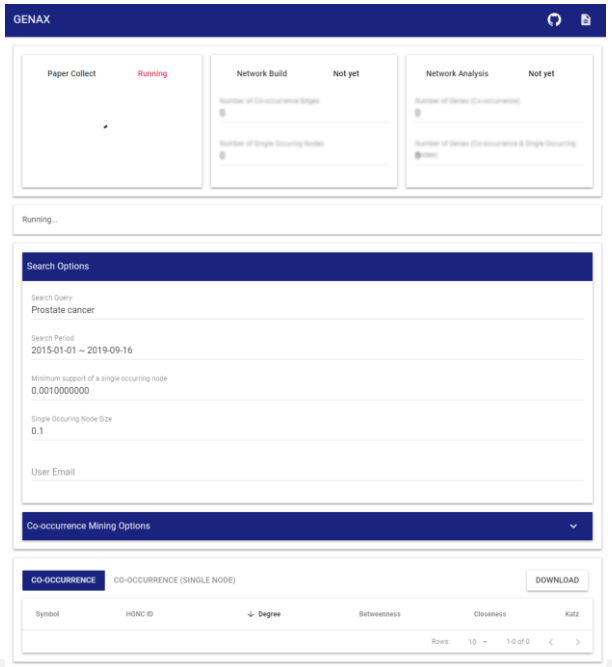
  - 질병에 관련 있는 Gene List를 검색할 수 있는 Web-based Service 프로젝트
    - Disease 검색 → Biomedical 논문들 → 데이터마이닝 → Gene-Gene 상관관계 추출 → Gene Ranking
    - Public data API 활용
      - PubMed
        - Biomedical 논문 search API
      - HGNC
        - Gene recognition API

- GENAX
  - 본 프로젝트는 다음 3가지 서브프로젝트로 나뉨
    - Frontend Web Application
      - Vue.js
    - Backend Server
      - node.js
    - Engine
      - Crawler / Processor
        - Python
      - Analyzer
        - Weka(Java), Bash Script
      - Database
        - MySQL
  - Frontend / Backend / Crawler / Processor / Database / Docker 개발을 담당함

Works / Projects

- GENAX
  - Frontend Issue
    - 검색 UI
      - 문제: 검색 UI에 Field가 너무 많음
      - 해결: Step by Step UI를 구성하자!
        - Stepper UI를 구성하여 사용성을 늘림
        - 필수 Field가 들어가지 않으면, 진행이 되지 않도록 구성
        - State 관리는 Vuex로~
        - CSS…를… 구겨…넣자…
    - 결과 UI
      - 문제: 분석 Engine이 돌아가는 모습을 실시간으로 보여주자!
      - 해결: 서버에서 Push Request를 발생시켜주면 안될까?
        - 서버가 Pull Request 기반으로 구성되어 있음
        - Push API로 연결시켜두면 좋겠지만…
        - 일단 시간이 없으니 Pull로 구성해놓고 나중에 고치자…
          - 결국 못 고침…





- GENAX
  - Deploy Issue
    - Deploy 과정이 너무 까다롭다
      - Frontend, Backend, Engine
      - 재부팅마다 linux screen을 생성해서 서버를 올려놔야 함
    - Docker를 써보자
      - Frontend는 Dockerize가 손쉽게 가능함
      - Backend, Engine은 힘든 상황
        - Why? MySQL이 Host에 구축되어 있음
          - Dockerize를 결정했을때, 이미 많은 데이터가 저장되어 있었음
          - MySQL 의존도가 그나마 낮은 Backend는 Dockerize
            - Network 옵션을 “Host”로 설정함
          - MySQL 의존도가 높은 Engine은 그대로 Local에서 실행
    - docker-compose로 정리하여, 한방에 실행되도록 구성
      - docker-compose up –d
      - Engine은 그대로 linux screen을 통하여 실행

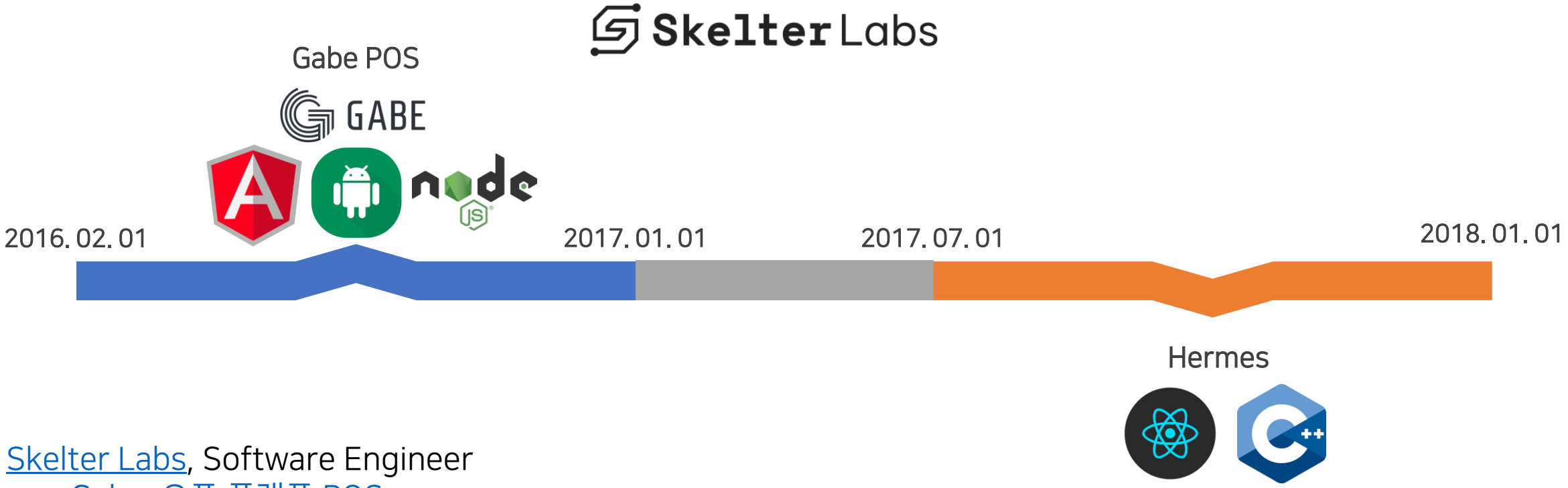
  - Web Link
    Github Link
    Paper Link (Engine Part)

- DELAB Homepage
  - Lab Homepage Renewal Project (Web)
  - 개요:
    - 기존 연구실 홈페이지 관리를 담당하고 있었음
      - Apache2, Php, MySQL
      - 멤버, 뉴스 관리 정도만…
    - 2018. 05월 경에 홈페이지가 갑자기 접속되지 않음
      - 문제 파악 시도
        - php 버전이 문제?
          → php7.2로 굉장히 구 버전을 사용하고 있었음
          → 어찌 어찌해서 php7.2로 재설치해도.. 해결이 안됨..
        - Apache2가 php를 제대로 못 뿌려주고 있나?
          → /etc/apache2 설정 확인해 봄
          → 어떻게 설정되어있는지 확인조차 불가 (무수히 많은 Legacy 설정들, 난잡한 파일 구성)
          → conf 파일을 조율하여 php 로딩까지는 살려냈으나, css가 안먹는다…
      - 그냥 새로 홈페이지를 만드는게 더 빠를거라 판단
        - 흔한 사용 문서조차 없는 코드
        - A를 고치면 B,C,D,…가 터짐 → 그냥 A를 안 고침…

- DELAB Homepage
  - 구현:
    - 기존 홈페이지의 문제점
      - Apache2 / Php / MySQL이 너무 옛날식이다
        - 접근성이 좋은걸로다가 해보자
      - Code 유지보수가 너무 힘듦
        - UI는 Php, 데이터는 MySQL… 굳이 이렇게까지 해야하나
        - 데이터 하나 수정하는데 MySQL 테이블 구조를 다 파악하고 있어야 함
    - Jekyll을 써보자
      - Blog 용도로 많이 사용하지만, Homepage 용도로도 많이 사용중
      - Data를 Static하게 구성하므로, 별다른 DB가 굳이 필요하지 않음
        - 여기에 큰 단점이 존재함
          - 데이터 양이 많아지면, 홈페이지의 규모가 너무 커진다
          - 보안이 너무 극심하게 떨어짐 → 코드를 공개할 수 없어…
        - 하지만 기존 홈페이지의 문제점이 더 큰 문제점이라고 판단함
          - 보안은 github private repo를 쓰거나, 내부에서 구성된 gitlab을 활용하자
          - email은 한번 protect를 걸어서, crawling에 안 잡히게!
          - 생각보다 데이터 양이 그렇게 많지 않다! (논문 데이터, …)
      - 유지보수
        - 그냥 markdown만 작성하면 해결되도록 구성하자

- DELAB Homepage
  - 효과:
    - 홈페이지 Develop, Build, Test, Deploy가 쉬워졌다!
      - **Develop**: Markdown만 작성
      - **Build**: Jekyll build command 사용
      - **Test**: Jekyll serve command 사용
        - WEBrick Serving하여 바로 local에서 확인할 수 있음
      - **Deploy**: 특정 폴더에 build하고 Apache2로 Serving
        - Apache2는 그냥 build 폴더를 Serving하면 됨
        - Conf 파일 간소화
    - 서버가 고장나도, 홈페이지를 다른 서버에 바로 살릴 수 있음
      - Github private repo로 관리
      - Jekyll, Bundle만 버전에 맞게 설치한 뒤 plugin 설치하면 바로 사용 가능
      - Apache2 Conf Back up 파일을 따로 저장
      - 설치 과정을 문서화로 정리
  - Web Link

Works / Projects



- Skelter Labs, Software Engineer
  - Gabe, 오픈 플랫폼 POS
    - Web (AngularJS)
    - Android (Java)
    - Server (node.js)
  - Hermes, Chatbot assistance
    - Web (React)
    - Engine (C++14)

Skelter Labs

- Gabe POS
  - 매장용 POS 서비스
    - 2016.02 ~ 2017.01
    - Software Engineer
    - 언어 / 프레임워크
      - Android(Java), Web(AngularJS), Server(node.js)

  - POS의 Inventory 시스템을 개발함. (node.js, AngularJS)
    - 관리자 Web 페이지
    - Inventory 관리 서버 (구입 주문 기록 구현, 판매에 따른 재고 계산 구현)

  - POS application을 자동으로 테스트하는 시스템 개발. (node.js, Appium)
    - End2End Test를 Appium을 사용하여 개발
    - Monkey Test

  - POS의 Localization 지원 (node.js, AngularJS, Android)
    - Localized POS Calculator 개발
      - ex) 1,000원 = $1.000
      - KRW는 ","으로 표기, USD는 "."으로 표기
      - KRW는 뒤에 '원'을 표시, USD는 앞에 '$'를 표시

Skelter Labs

- Gabe POS
  - Currency 별 절사 시스템 개발 (베트남 사업 대비)
    - ex) 베트남의 Currency인 '동(VND)'
      - 베트남 동은 거래를 할때는 1000단위로 거래하나, 수수료나 거스름 돈에서 100단위의 잔돈이 발생
      - 베트남 동은 100 아래로는 의미가 없으므로, 반올림 / 올림 / 내림을 할지 고민함
      - 현지 자료, 리포트, 논문 등을 찾아보며 현지에서 자주 사용하는 방식으로 해결함
      - 거스름돈은 올림, 수수료는 내림으로 처리함
  - POS Android 앱을 리뉴얼 (Android)
    - UI / UX Changing
    - Android 6.0, Material design을 따르도록 UI 구현
      - Bottom Bar Navigator
  - POS Backoffice 웹페이지를 리뉴얼 (AngularJS)
    - Angular Material을 사용하여 Material design을 따르도록 UI 구현
  - QR Code Generation (JS)
    - 영수증의 QR Code를 생성하는 로직 구현
    - qrcodejs library 사용

Skelter Labs

- Hermes, Chatbot assistance
  - Chatbot Assistance 서비스
    - 2017.07 ~ 2018.01
    - Software Engineer
    - 언어 / 프레임워크
      - React, C/C++

  - Chatbot engine의 session 시스템을 개발함. (C++)
    - User가 사용한 Bot마다 각기 다른 state를 저장하도록 구현
      - SESSION_ID = (User, Bot)
    - Thread-safety 고려
      - 동시에 같은 유저가 한가지의 Bot에 대해서 Get / Set을 요청할 수 있으므로 Get / Set 모두 Locking이 필요함
      - Mutax(Google Lock)를 사용하여 Thread-safety 보존

  - Chatbot Backoffice 관리자 페이지를 개발 (React)
    - Chatbot을 프로그래밍을 모르는 비개발자라도 쉽게 구현할 수 있도록 개발
    - React, Redux, React-Router 등을 이용하여 구현
      - Why React?
        - Backoffice에서 Chatbot을 시연할 수 있는 UI가 필요하므로, UI가 복잡한 편에 속함
        - 복잡한 UI에서는 React-Redux가 UI State 관리에 출중하므로 React를 선택함
    - 사내 React Seminar 진행 (PPT Link)

- Etc Projects
  - Time It(타임잇), 시간관리 앱
    - 2014. 06 ~ 2015. 06
    - 연세대학교 정보통신연구실 인턴
    - 언어 / 프레임워크
      - iOS7, Swift 1.2, SQLite3
      - Cocoapods
    - 시간 관리 시스템 및 UI 개발, 스타트업 경험

  - LG G6: Ergonomic Evaluation of Smartphone Usability Test Application
    - 2016.12 ~ 2017.01
    - 연세대학교 상호작용설계연구실 외주
    - 언어 / 프레임워크
      - Android 6.0
    - Smartphone Usability를 test & logging 하는 android application을 외주 개발함
      - 스마트폰 화면 크기에 따른 사용성 테스트
    - Link

2. Papers

- Journal
  - International
  - Domestic
- Conference
  - International
  - Domestic

Papers

- Journal
  - International
    - IMA: Identifying disease-related genes using MeSH terms and association rules
      - Jeongwoo Kim, Changbae Bang, Hyeonseo Hwang, Doyoung Kim, Chihyun Park, Sanghyun Park
      - Journal of Biomedical Informatics, Dec 2017, Volume 76, 110-123 pp.
      - Link

  - Domestic
    - 인-메모리 키-값 데이터베이스를 위한 비 휘발성 메모리 기반 영속적 로그 버퍼 기법
      - 김도영, 최원기, 성한승, 이지환, 박상현
      - 정보과학회논문지, Nov 2018, Volume 45, Number 11, 1193-1202 pp.
      - Link

2 Papers

- Conference
  - International
    - A Scalable and Persistent Key-Value Store Using Non-Volatile Memory
      - Doyoung Kim, Won Gi Choi, Hanseung Sung, Sanghyun Park
      - The 34th ACM/SIGAPP Symposium On Applied Computing(SAC), Limassol, Cyprus, Apr 2019.
      - **Best Poster Paper Awards** (Link)
      - Link

  - Domestic
    - 디스크 기반 키-값 데이터베이스의 SST 파일에 대한 GPU 가속화 필터 기법
      - 김도영, 최원기, 노홍찬, 박상현
      - Korean Computer Congress 2019 (KCC2019), 제주, 대한민국, Jun 2019.
      - **Best Paper Presentation Awards** (Link)
      - Link
    - 비휘발성 메모리를 이용하여 데이터 영속성을 유지한 인 메모리 키-값 데이터베이스
      - 김도영, 박상현
      - Korean Computer Congress 2018 (KCC2018), 제주, 대한민국, Jun 2018.
      - **Best Paper Awards** (Link)
      - Link

3. Educations

- Master
  - 연세대학교 컴퓨터과학과 석사
    - 데이터공학연구실 (Link)
  - 2018.02. ~ 2020.02. 졸업 예정
  - 전체 평균 학점: **4.27 / 4.3**
    - 전공 평균 학점: **4.3 / 4.3**

- Bachelor
  - 연세대학교 컴퓨터과학과 학사
  - 2012.02. ~ 2018.02.
  - 전체 평균 학점: **3.77 / 4.3**
    - 전공 평균 학점: **4.16 / 4.3**

4. Awards

- GPA
  - 연세대학교 학점 우수상 (2014. 3학년 1학기)
  - 연세대학교 학점 우수상 (2013. 2학년 2학기)
  - 연세대학교 학점 최우수상 (2013. 2학년 1학기)

- Paper
  - Best Paper Present Awards
    - 한국정보과학회 한국컴퓨터종합학술대회, KCC2019
  - Best Poster Paper Awards
    - SAC 2019
  - Best Paper Awards
    - 한국정보과학회 한국컴퓨터종합학술대회, KCC2018

5. About Me

- Military Service
  - Unfulfilled (전문연구요원 신규 편입 필요)
- Job
  - Can do right now (Familiar)
    - DB-Engine(Redis/RocksDB), CUDA, Web(Vue.js)
  - Can do with learning / following
    - Web(React), Server(node.js), Android(Java), Spark, Docker
  - Want to learn
    - DevOps(Docker, k8s, clouds), Android(Kotlin/Flutter), Serverless, ML/DL(TF, PyTorch), ML/DL Infra

5. About Me

- Email
  - arbc139@gmail.com
- Github
  - Totorody
  - w4-dykim