국내 공개SW R&D 과제 사례
@제 3회 공개SW 거버넌스 아카데미

2017. 9. 21.

정성인, 차세대OS기초연구센터, ETRI

목차

- ‘매니코어 기반 초고성능 스케일러블 OS 기초연구‘ 과제 개요
- SW 개발 모델의 변화
- 공개SW 기반 R&D 수행시 고려사항
  - 기존 커뮤니티에 기술 피드백
  - 개발 후 공개
  - 오픈 프로젝트

참고문헌

- Producing Open Source Software : How to Run a Successful Free Software Project(By Karl Fogel)
- Open Source Compliance in the enterprise(By Ibrahim Haddad, Linux Foundation)

과제 개요

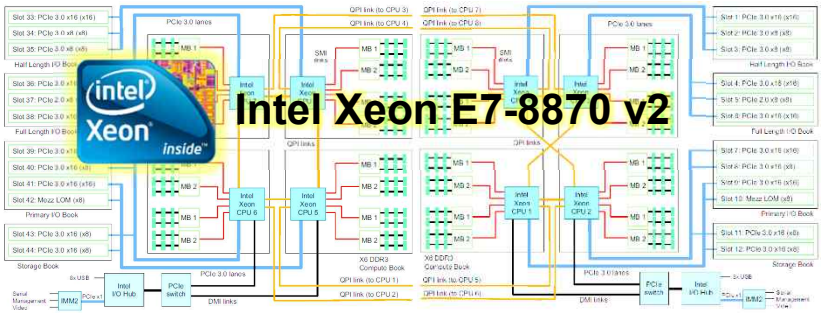
- 과기정통부 SW기초연구과제(2014 ~)
  - SW기초연구센터 과제(2개), 스타랩과제(16개), 차세정과제(10여개+)
- 매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대OS기초연구센터)
  - 2014-2021, 공개SW과제 1호
  - ETRI + 12개 국내외대학연구실 + 2개 기업 + 공개SW협회
  - 목표 : 코어 수 증가에 따른 OS 성능 증가(스케일러빌러티 제공)
  - 대상 기술 : 리눅스(모노리틱구조), 새로운 구조의 운영체제(멀티커널 구조), 병렬화 환경, 도구
  - 오픈소스 활동 방식
    - 리눅스 커뮤니티에 피드백
    - 개발 후 공개
    - 오픈 프로젝트
  - https://github.com/oslab-swrc(2017년 하반기에 공개)

개발 시스템

매니코어 리눅스 테스트베드

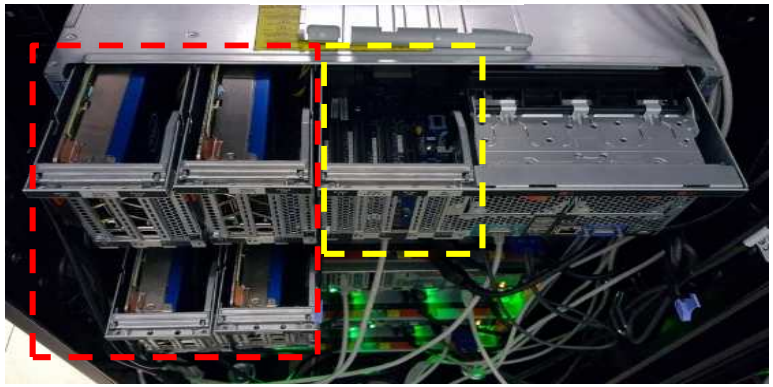
IBM x3950 X6

120코어(15 코어 * 8 소켓)



- IBM 192코어(24코어 * 8소켓)

멀티커널 운영체제 테스트베드



1,000코어 (120 + 220*4)

Intel Xeon Phi 7120P
- 220코어(55 코어 * 4 thread)

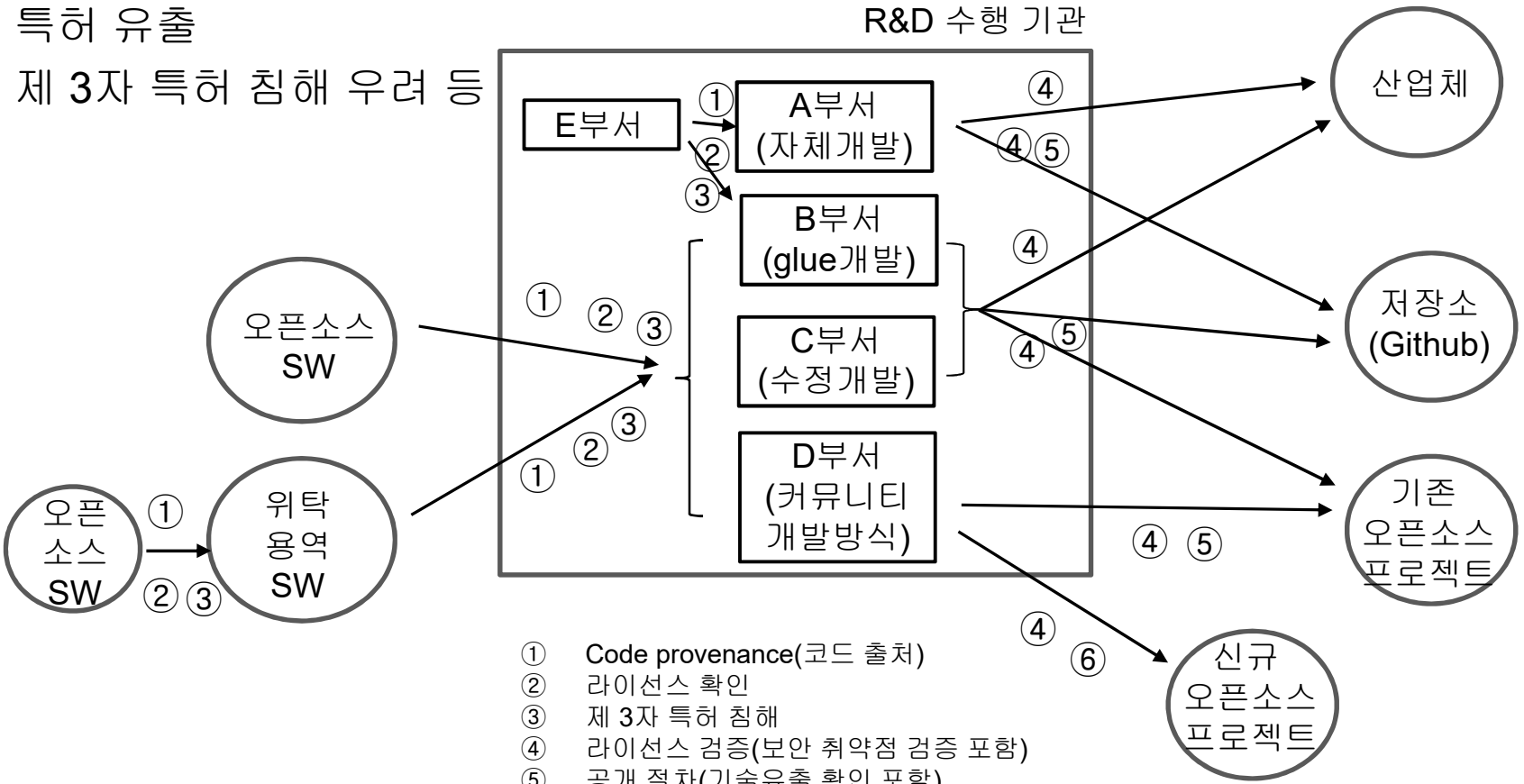
Intel NVMe SSD 750 Series
- 1.2 TB

SW 결과물

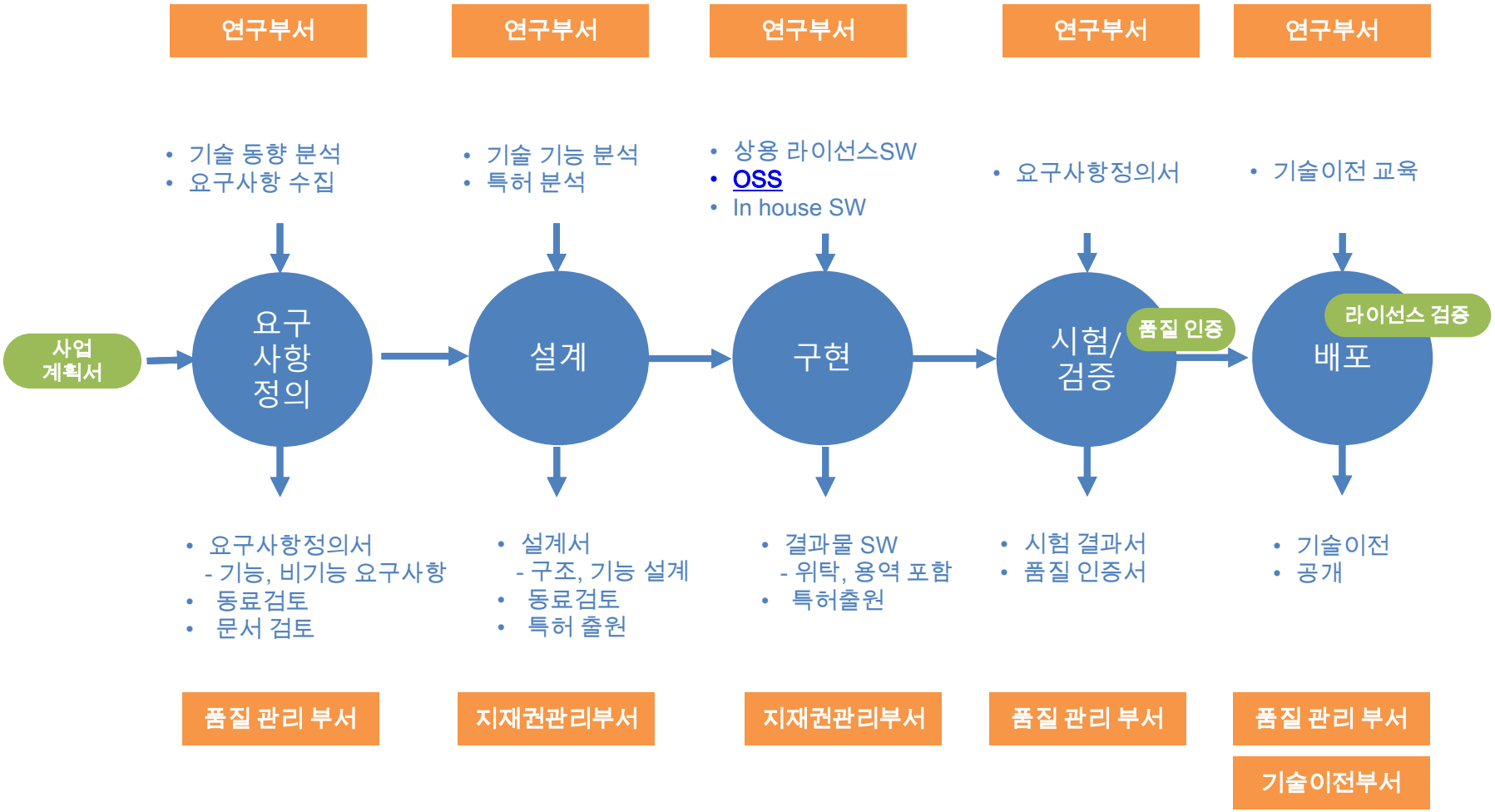
- Magnolia : 스케일러블 매니코어 리눅스
  - nvm2sfs : scalable file system
  - CST-lock : NUMA aware blocking sync
  - VM scalability : enhancing locking primitive on VM
  - ORDO : scalable timestamp-based data structure by invariant HW clock
  - LDU : lockless data structure
  - flsched : less fairness scheduler
  - N4 : multi-level scheduler
  - Core partition
- Solros : 리눅스기반 멀티커널 OS
- Azalea : 경량커널 기반 멀티커널 OS
- Mosaic : 대규모 그래프 프로세싱 엔진
- Juxta : cross-checking semantic 도구
- APIsan : API cross-checking semantic 도구
- Fxmark : 파일 시스템 벤치마크
- 스케일러블 프로파일러
- 스케일러블 JVM :
  - WASP : workload aware task scheduler and partitioner
  - JVM NUMA optimizer

SW 개발 모델 변화

- Multi-source development model
  - 라이선스를 고려하지 않는 개발 : OSS -> 상용SW, licensing chimera
  - 라이선스 미준수
    - Notice 등 의무사항
  - 특허 유출
  - 제 3자 특허 침해 우려 등

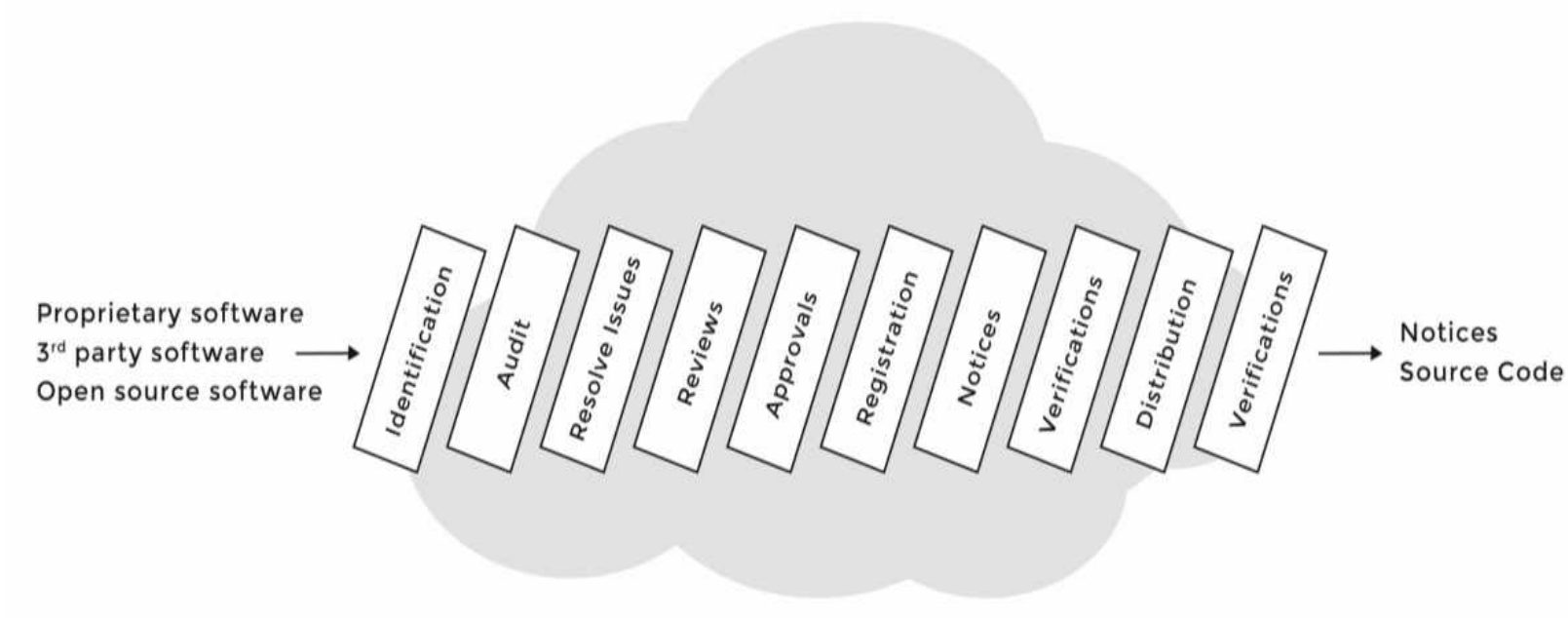


일반적인 연구개발 프로세스(SPICE, CMMI 준수)



오픈 소스 컴플라이언스 프로세스

- Open source compliance in the enterprise(Ibrahim Haddad, LF)



공개SW 기반 R&D 수행 시 고려사항

- 과제 초기 단계에 과제목적, 전략 등에 맞는 라이선스 선정
  - 사례, 리눅스가 성공한 이유?
  - 특허 전략
- 활용하는 OSS의 라이선스 사전 확인
  - 가능하면 제 3자 특허 포함 여부 확인
  - 코드 출처 확인
  - 라이선스 호환 확인
- SW 설계 시 기능적인 검증뿐만 아니라 라이선스 검증
  - 구조적 관점, linkage 관점 검증, 헤더 파일, 통신 방법, 커널 vs. 사용자 공간, …(architecture diagram)
  - 특허 출원
  - 사용하는 공개SW 등록(clearing house)
- 모듈화가 잘된 코드, 읽기 좋은 코드 등 활용하기 좋은 코드
  - 라이선스 notice
  - 불필요한 라이선스 관련 주석은 하지 않도록
  - SW 메타 데이터(SPDX)
- 주관, 참여, 위탁, 용역 기관간 라이선스 호환 등
  - 발주시, 라이선스 등 명확한 요구사항 명시
  - 검수시, 명시한 요구사항 확인 절차 수행
  - SPDX, Fossology, OpenChain 참조
- 관련 기술의 외부 개발자 활용
- 조직의 역할 분담
  - 개발부서, 지재권 부서, 컴플라이언스 부서

리눅스 커뮤니티에 기술 피드백

- 관련 내용
  - Most free software projects fail
    - Failure is not an event
    - There is not even a clear definition of when a project is expired
  - How to introduce a new free software project to the world
    - But first, look around
      - Might find something so close
      - Exceptionally, educational experience, pre-exiting code won't help
    - Found that nothing
      - Private vision into public vision
      - Every good work of software starts by scratching a developer's personal itch
- 과제현황
  - 222여개의 버그 패치 제출
  - 2개의 기능 패치 제출 -> 기술개념으로 accept되어 리눅스 lock 알고리즘에 적용
  - 파일 시스템 등 리눅스 커널 버그 패치(202개), 보안 버그 패치 제출(18개)
- 활동 시 고려사항
  - 라이선스보다 특허 침해 신중
  - 해당자에게 제출 및 관련 개발자 CC
  - 같은 개발 환경
  - 개발 이전에 해당자와 상의(early discussion, peer review) – 사례, pthread
  - 무한 경쟁체제 이해 – 사례, ETRI cgroup
  - 코드 스타일
  - 충분한 실험 및 결과 제시, 논문을 같이 제출
  - 패치 제출 방식 준수
  - GPL-compatible code
  - 코드에 제 3자 특허, 자신의 특허 포함 여부(기여자 계약)
  - SW 지재권 등록(기여, 공개하더라도)

개발 후 공개

- 과제 현황
  - 해당 오픈 프로젝트가 없는 경우
  - 제안 기능을 accept하지 않은 경우
  - 논문발표 후 SW를 공개해 두는 경우
    - SW기초 연구과제 경우, 기술의 개념 정립 -> 공개 -> 오픈 프로젝트 과정 적절
- 고려 사항
  - Technical debt
  - 라이선스 : 과제의 목적, 공개하는 목적, 특허 등 여러 고려사항 후 선정
    - 공개하고 특허 확보 vs. 모두 공개 판단
  - 특허 관련 내용 고지
    - 명시적으로 특허 무상 허여(GPLv3, MPL, EPL, Apache2.0)
    - 명시적 특허 허용은 아니지만, 묵시적으로 특허 허여(GPLv2)
    - 특허에 대한 어떤 언급이 없는 경우,묵시적 허여 추정(BSD, MIT)
  - 제 3자 특허 포함? : 허여 또는 대체 기술 이용
  - SW 저작권, 특허 등록(공개하더라고)
  - 기술 개요, 사용(설치, 컴파일,..)방법, to-do 리스트, 연락처 등 기술 정보 제공
  - 소스코드에 개발자 이름?(cookie licking)
  - 기존 오픈소스를 기반하여 개발한 것을 공개하는 경우, 해당 라이선스 고지 의무 준수
  - 홍보

오픈 프로젝트

- 과제 현황
  - 집단지성이 필요한 분야
  - 이어서 추가 개발이 필요한 분야
  - 2~3개 고려
  - Who? 과제참여자 vs. 외부 개발자
- 고려 사항
  - 프로젝트 이름 작명
  - 무슨 목적의 프로젝트인지 명확히 제시(clear mission statement, 방문자가 30초만에 인지하도록)
  - 첫 페이지에 명확한 라이선스 정보 제공
  - 기술 특징 및 요구 리스트
    - 완벽하게 개발된 소프트웨어?
  - 개발 현황
  - 다운 로드 링크
  - 버전 관리 및 버그 tracker : A sign that please join and help
  - 커뮤니케이션 툴 : 메일링 리스트, IRC, forum, SNS -> 사례, 13세 개발자
  - 개발자 가이드라인 등 문서들, SW 시연물(screenshots, videos, and example output)
  - Canned 사이트(예, github)
  - 홍보
  - 거버넌스 : 의사결정 프로세스, 선량한 독재자 모델 vs. 능력주의 모델, 기술적 사항 -> 조직 및 운영 사항, 개방적이고 간단하게 마련(예, 심비안 오픈 프로젝트 실패)
  - 지재권 관리 : 라이선스, 기여자 계약(포괄적인 허락 받음, 라이선스 변경, 상용 라이선스로 배포 등), 상표권 관리(인지도, 스폰스쉽 -> fork 방지 등 목적)
  - SW 패키징, 릴리스 계획 : Highway repair, 릴리스 기간에 갈등
  - 개발 참여자의 보상(credits)
  - 관련 기술의 다수의 경쟁업체 참여 -> 생태계 구성
  - 오픈 프로젝트의 fork에 주의

감사합니다.