

HL-LHC ATLAS ピクセル検出器量産時の品質試験に向けた データベースシステムの構築

東京工業大学 理学院物理学系物理学コース 陣内研究室
奥山広貴 (19M00398)

2020年12月17日

概要

第1章

LHC-ATLAS 実験と検出器アップグレード 計画

1.1 LHCについて

LHCとATLAS実験は分けたほうがいい。

1.2 ATLAS実験

1.2.1 内部飛跡検出器

1.2.2 カロリメータ

1.2.3 ミューオン検出器

1.3 HL-LHC実験アップグレード計画

1.3.1 加速器アップグレード

1.3.2 内部飛跡検出器のアップグレード、現行との違い

1.3.3 期待される物理

第2章

新型ピクセルモジュール

2.1 シリコン検出器

2.1.1 半導体

2.1.2 pn接合

2.1.3 検出原理

2.1.4 放射線損傷

2.2 プロトタイプピクセルモジュールの構成

2.2.1 要求

2.2.2 シリコンセンサー

2.2.3 読み出しフロントエンドチップ

2.2.4 PCB

2.2.5 モジュールキャリア

2.3 モジュールの種類

Quad, triplet

2.4 現行ピクセルモジュールとの違い

第3章

検出器量産と品質試験

3.1 検出器量産

3.2 組み立て工程

3.3 品質試験

各組み立て工程に対して、いくつかの品質試験を行う。行う品質試験の代表的なものを以下に示す。

- 外観検査
- 質量測定
- 平坦性測定
- 電気的試験

各試験項目についての詳細と現時点での決定事項を以下に記す。

3.3.1 外観検査

3.3.2 質量測定

3.3.3 平坦性測定

3.3.4 電気的試験

電気的試験はさらに以下のようないくつかの項目に細分化される。

First Power Up

センサー特性確認試験

SLDO VI

Chip Configuration

読み出し試験

Bump bond quality

3.3.5 簡易電気的試験

3.4 品質試験に用いるソフトウェア

3.4.1 YARR

3.4.2 QC helper

第4章

モジュール情報及び品質試験結果管理システム

前章で述べたように、モジュール生産及び品質試験を世界中で行う。これらの情報はデータベースシステムを用いて管理することが決まっている。現在この開発を行っている。システムについては、大きく2つに分けられる。チェコに設置し、試験運用をしている中央データベースと、各組み立て期間に設置し、運用の際に使用するローカルデータベースである。本章ではこれらのデータベースについて説明する。また、システム開発の中で私が開発を行った仕組みや機能について詳細に説明する。

4.1 中央データベース

4.1.1 中央データベースの概要

概要

中央データベースは、新型内部飛跡検出器の製造に関する全ての情報の保存を目的として開発されたデータベースである。ユニコーン大学が開発、運用を行っていて、チェコにデータベースサーバーが設けられている。新型内部飛跡検出器は、前述したようにピクセル検出機とストリップ検出機にから構成される。これらを生産するにあたって、シリコンセンサーや電気基板といった小さな部品から製造を行い、それらを用いたモジュールの組み立て、複数モジュールを搭載した stave や ring の組み立てを経て検出器が完成する。また各組み立て段階において、動作確認等を目的とした品質試験を行う。これらの過程における全ての構成部品の情報、及び品質試験結果を中央データベースに保存する。

意義

中央データベースに保存された情報は、検出器運転時の参考値として扱われる。モジュールを例にだと、品質試験で読み出し試験を行った際の最適な設定値を中央データベースに保存するため、実際の運転時に参照することができる。また運転前の状態における検出器の性能、運転前後での検出機性能比較を行うことができる。HL-LHC では1章で述べたように、運転時における放射線量が大きいものとなるため、運転前後での放射線損傷の影響の研究を行うことができ、検出機の寿命の推定や放射線損傷に関しての対策に役に立てることができる。

それらしいデータかこんな解析に役立てたいみたいなのを考え中です。

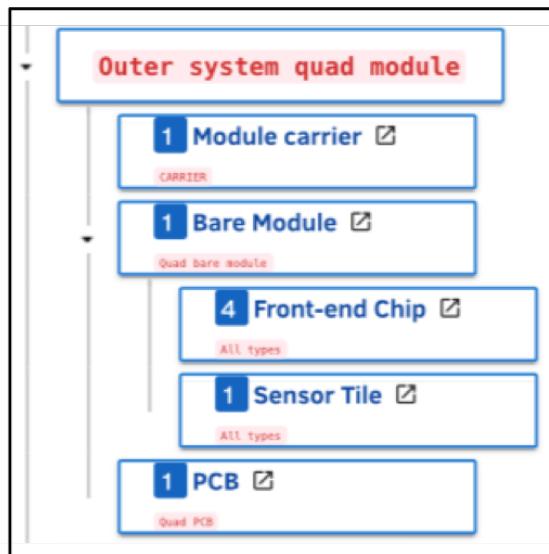


図 4.1 モジュール構造の一例

4.1.2 モジュール情報構造および構成部品との関係の実装

中央データベースにモジュールを登録するためには、1章で述べたようにモジュールの種類、モジュールを構成する部品といった情報構造を決定し、データベース上に定義しておく必要がある。この情報構造をデータベースに実装し、登録できる仕組みを整えた。詳細な種類と構造については表??に示す。またある種類に関する例を図??に示す。

4.1.3 組み立て工程および品質試験の情報形式の実装

モジュールの情報構造の実装に加えて、品質試験の情報を正確に管理するには、モジュール組み立て工程の情報と付随する品質試験の項目をデータベース上に定義する必要がある。これを実装し、テスト結果を適切な組み立て工程へアップロードできる仕組みを整えた。

詳細な構造を図??に示す。

表4.1 中央データベースにおけるモジュールの種類と構造

種類	構成する部品(数)
Triplet L0 stave module	Single bare module(3) Triplet stave PCB(1)
Triplet L0 Ring0 module	Single bare module(3) Triplet R0 PCB(1)
Triplet L0 Ring0.5 module	Single bare module(3) Triplet R0.5 PCB(1)
L1 quad module	Quad bare module(1) Quad PCB(1)
Outer system quad moudle	Quad bare module(1) Quad PCB(1)
Outer system quad moudle	Dual bare module(1) Dual PCB(1)
Digital triplet L0 stave module	Digital single bare module(3) Triplet stave PCB(1)
Digital triplet L0 Ring0 module	Digital single bare module(3) Triplet R0 PCB(1)
Digital triplet L0 Ring0.5 module	Digital single bare module(3) Triplet R0.5 PCB(1)
Digital quad module	Digital quad bare module(1) Quad PCB(1)
Digital L1 quad moudle	Digital quad bare module(1) Quad PCB(1)
Dummy triplet L0 stave module	Dummy single bare module(3) Triplet stave PCB(1)
Dummy triplet L0 Ring0 module	Dummy single bare module(3) Triplet R0 PCB(1)
Dummy triplet L0 Ring0.5 module	Dummy single bare module(3) Triplet R0.5 PCB(1)
Dummy quad module	Dummy quad bare module(1) Quad PCB(1)
Dummy L1 quad moudle	Dummym quad bare module(1) Quad PCB(1)

表4.2 中央データベースにおける組み立て工程と付随するテスト項目

組み立て項目	付随する組み立て情報及び品質試験項目
1. Bare to PCB assembly	Visual Inspection Metrology Mass measurement Glue information
2. Wirebonding	Visual Inspection Wirebond information (Wirebond pull test) First power up Sensor IV SLDO VI Chip configuration Pixel failure test
3. Wirebond Protection	Visual Inspection Potting information Sensor IV Chip configuration Readout for basic electrical
4. Parylene Coating	Visual Inspection Parylene information Mass measurement Sensor IV Chip configuration Readout for basic electrical Bump bond quality
5. Thermal Cycling	Visual Inspection Thermal cycling info Sensor IV Chip configuration Readout for basic electrical
6. Burn-in	Visual Inspection Metrology Mass Measurement First power up Sensor IV SLDO VI Chip configuration Pixel failure test
7. Reception	

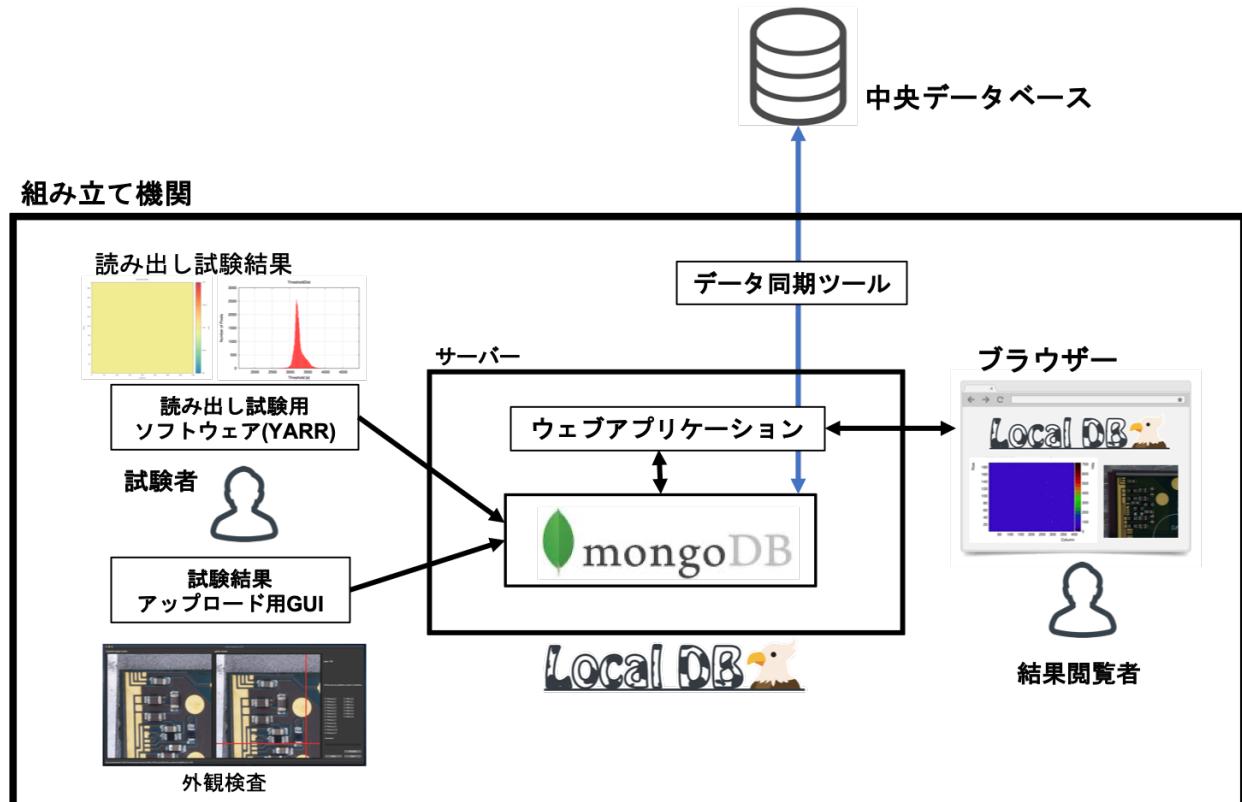


図4.2 ローカルデータベースシステムの概要

4.2 ローカルデータベース

4.2.1 ローカルデータベースの概要と意義

中央データベースでは、前述したようにモジュールの情報のみならず新型内部飛跡検出器に関わるすべての情報を管理する。データベースの機能としては汎用的に使えるようなものになっている。モジュールの組み立て及びその品質試験に関しては3章で述べたように工程が複数に渡り、行う品質試験の数も多い。1つの生産現場で多いところでは数千個のモジュールを作ることになるため、データ管理が簡単にかつ円滑に進むようになっているのが好ましい。このような理由から、生産現場での生産性、利便性に特化し、円滑な生産をサポートすることを目的としたデータベースシステム(ローカルデータベース)を開発している。システムの概要図を図??に示す。オープンソースのサービスであるMongoDBを各生産現場で使い、開発したウェブアプリケーションを併用することでデータ管理や中央データベースとのデータ同期を行うシステムとなっている。

具体的にローカルデータベースは以下のようない点を持つ。

- ローカルにデータベースサーバーを立てるためアクセス速度が早く、円滑にデータ管理を行うことができる。
- モジュールの組み立て工程を管理し、生産者の適切な処理を助ける。
- モジュールに特化したデータ管理、解析を行うことで異常をいち早く検知できる。
- 試験者の情報や試験時間など、テスト結果以外の必要な情報を正確に管理できる。

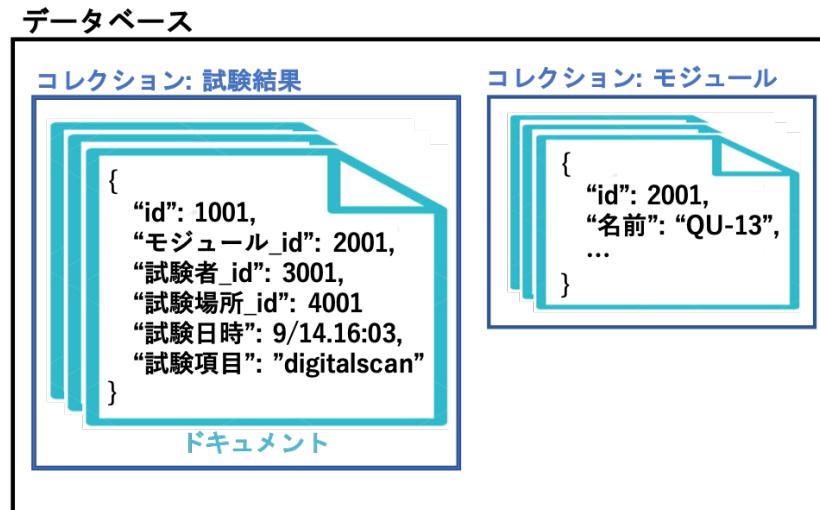


図 4.3 MongoDB の構造の例

4.2.2 MongoDB と内部構造

MongoDB とは NoSQL に分類されるデータベースである。MongoDB の構造について簡単に表したものを見た。一般的な SQLDB のようにテーブル形式ではなく、JSON 形式で情報を格納する。情報を保持している一枚のシートを「ドキュメント」と呼び、「コレクション」と呼ばれる枠に複数のドキュメントが格納されている。各ドキュメントは ID を持っていて、異なるコレクションにおけるドキュメント間の紐付けはこの ID を用いて行う。

ローカルデータベースシステムにおいて、MongoDB を使用する主な利点を以下に示す。

- 各コレクションに格納するドキュメントの構造が動的であるため、開発を柔軟に行うことができる。
- JSON 形式のため情報取得の際に変換処理が不要で、ウェブアプリケーションとの親和性が高い。
- 全ての処理をメモリ上で実行するため、高速な読み書きが可能。

モジュール及び品質試験に用いる主なコレクションと内部情報を表??に示す。

4.2.3 データ同期ツール

モジュールや品質試験の結果のデータ共有のために、中央データベースとローカルデータベースの間でデータ同期が行われる必要がある。これを行うツールを Python を用いて開発した。現在は以下の機能を実装している。

- モジュール ID のダウンロード
- 読み出し試験結果のアップロード

実装した開発項目について、詳細については以下で述べる。読み出し試験以外に実施する品質試験結果のダウンロード、アップロードの機能や、組み立て工程情報の取得等の機能が今後の開発課題として上げられる。

表 4.3 品質試験に用いる主なコレクション

データベース名	コレクション名	情報
localdb	component	モジュール情報、FE チップ情報
	childParentRelation	FE チップとモジュールの関係性
	QC.module.status	各モジュールに対する組み立て工程及び選択された試験結果
	QC.result	品質試験結果
	testRun	読み出し試験結果
	user	読み出し試験実施者
	institute	読み出し試験実施場所
	componentTestRun	component と testRun の関係性
	comments	コメント情報
localdbtools	QC.status	組み立て工程及び試験項目
	viewer.user	登録ユーザの情報
	viewer.query	読み出し結果キーワード、検索機能実行時に使用
	viewer.tag.docs	モジュール及び試験結果に付けるタグ情報

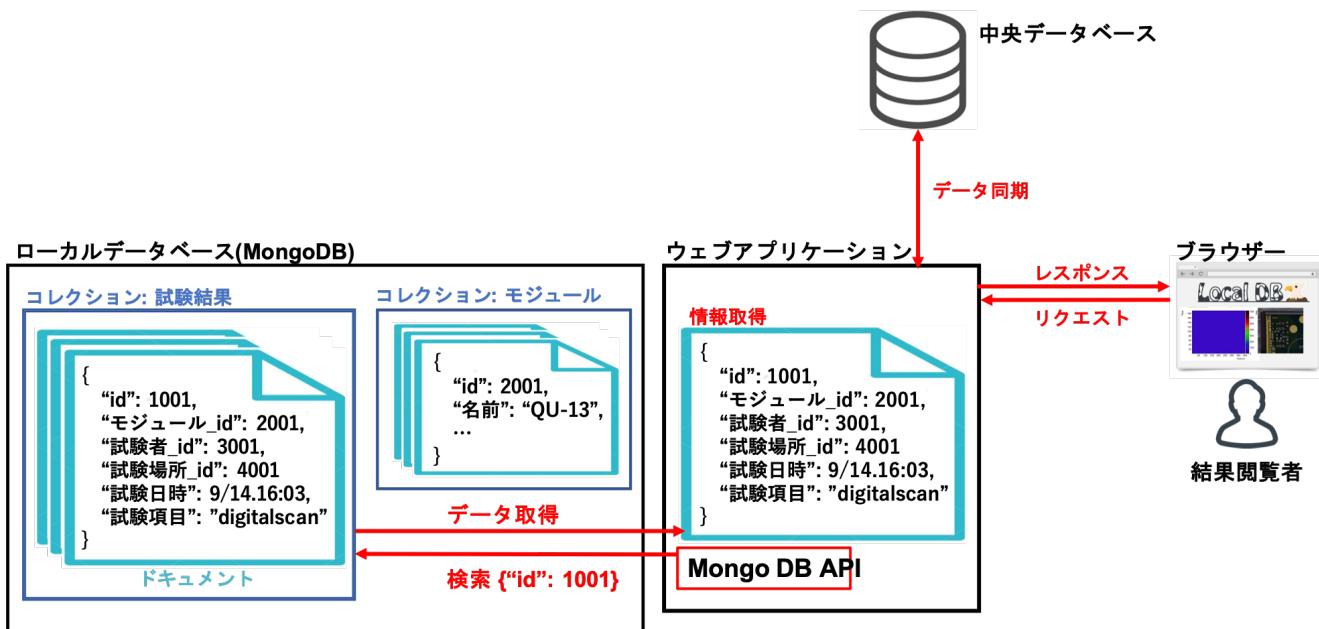


図 4.4 ウェブアプリケーション処理のイメージ

4.2.4 ウェブアプリケーション

各組み立て機関において、試験者が品質試験結果を閲覧、管理するツールとして、ウェブアプリケーションを開発している。ローカルデータベースとアプリケーション間の処理に特化したイメージを図??に示す。このようにアプリケーションはデータベースとブラウザー、データベース間のインターフェースとなっている。

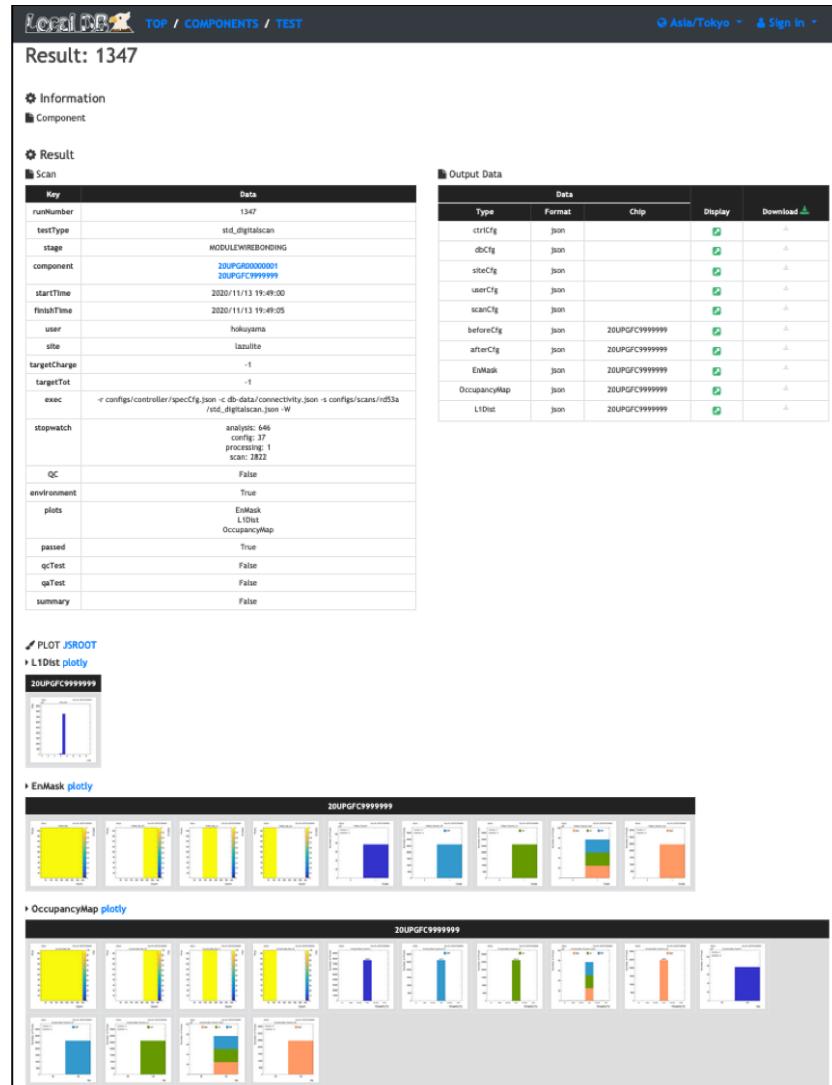


図 4.5 品質試験結果の例

試験結果を迅速に分かりやすく見るシステムを作り、円滑な生産の補助や異常結果の早期発見がアプリケーション目的としている。またデータベースの情報管理のみならず、閲覧上述したデータ同期ツールや、後述する試験結果の解析ツールなどの外部スクリプトの実行、結果取得といった、生産時における全てのユーザの操作はこのアプリケーションを用いて行う。

ウェブアプリケーションでは、現在以下の機能を使用することができる。ある品質試験の結果ページを図??に示す。

- 登録モジュール情報及び品質試験結果の閲覧機能
- ローカルデータベースにおけるユーザ管理機能
- 上述したデータ同期処理実行機能

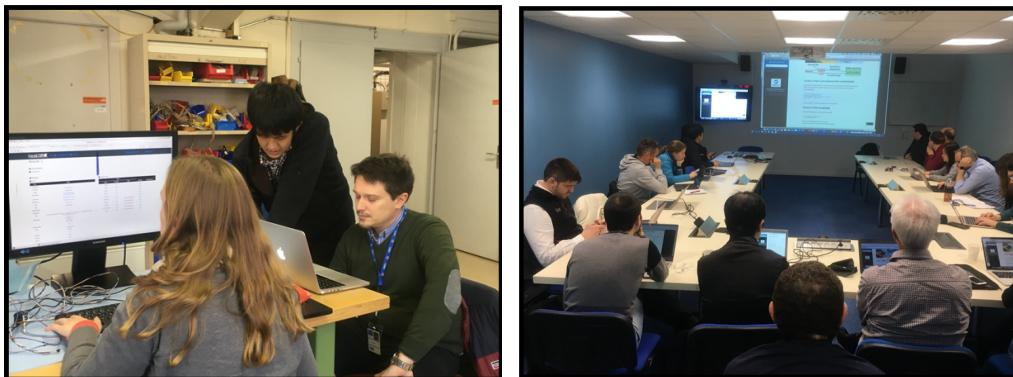


図 4.6 ハンズオンとハンズオフ

4.3 チュートリアルと普及状況

ローカルデータベースの機能の普及を目的として、2020年2月にCERN研究所にてシステムのチュートリアルを行った。このチュートリアルは以下のような2つのセッションに分けて行った。

- 参加者が実際にサーバーの設定、各ソフトウェアのインストールを行いながら機能を実践するセッション（2月3日から6日まで）
- 私が参加者の前で実際に機能を実践し、システムや使い方に対して議論を行うセッション（2月7日）

それぞれのセッションの様子を図??に示す。数多くの議論を行い、有益なフィードバックを得ることができた。また品質試験の流れにおいて、一連の機能確認をすることができた。

これを経て現在ローカルデータベースは世界10箇所にて導入され、試験運用が開始している。また将来的には全組み立て機関で使うことが決定しており、それに向けたシステム開発、サポートが必要となっている状況である。ローカルデータベースについて、導入及び試験運用を行っている機関を以下に示す。また世界地図を??に示す。

- 高エネルギー加速器研究機構(KEK), 日本
- 欧州原子核研究機構(CERN), スイス
- University of Liverpool, イギリス
- University of Oxford, イギリス
- University of Glasgow, イギリス
- Paris-Saclay University, フランス
- パリ第6大学, フランス
- フランス国立科学研究中心, フランス
- University of Grenoble, フランス
- University of Gottingen, ドイツ
- University of Siegen, ドイツ
- University of Genoa, イタリア
- University of Salento, イタリア

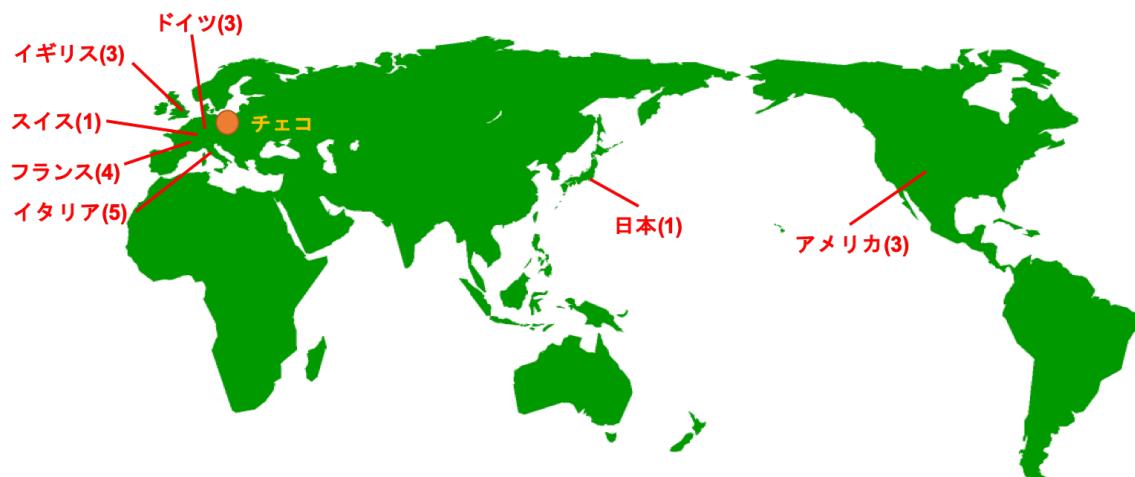


図 4.7 ローカルデータベース地図

- University of Milan, イタリア
- University of Udine, イタリア
- Univerity of Trento, イタリア
- University of Oklahoma, アメリカ
- Argonne National Laboratory, アメリカ
- Lawrence Berkeley National Laboratory(LBL), アメリカ

表 4.4 ユーザ権限一覧

ユーザ	付加される権限	使用できる機能
管理者	ユーザ管理権限 データベース読み書き権限 ウェブアプリケーションログイン権限	権限付きユーザ登録機能
権限付ユーザ	データベース読み書き権限 ウェブアプリケーションログイン権限	試験結果のアップロード 中央データベースとのデータ同期機能 その他ウェブアプリケーションの機能(コメント、タグ)
一般ユーザ		モジュール情報及び試験結果の閲覧

4.4 実装した機能

私は、このシステムの中に以下のような機能を実装した。詳細について以下に述べる。

- ユーザ管理機能
- 品質試験結果の登録と組み立て工程の管理機能
- 読み出し試験結果検索機能
- 中央データベースとのデータ同期ツール

4.4.1 ユーザ管理機能及び各種機能

異常があった際に確認することを目的として、誰が試験を行ったかを記録することが必要である。また、モジュールの登録や中央データベースとのデータ同期など、データベースの機能使用を制限することも必要である。これらを目的として、試験者及びデータベース使用者情報の管理システムを開発、実装した。この詳細について以下に述べる。

機能概要

データベース権限の段階として、管理者、権限付きユーザ、一般ユーザの3段階を設けた。各ユーザが使うことのできる機能を表??に示す。

権限付きユーザの機能としてモジュール及び試験結果にコメント、タグをつける機能を実装した。使用したときの様子を図??、??に示す。

ユーザ登録操作

機能の仕組み

ユーザ登録の際には内部で以下の2つの処理が行われるように実装した。

1. mongoDB アカウントの作成、読み書き権限の付与
2. ウェブアプリケーションで用いるユーザ情報ドキュメントの作成

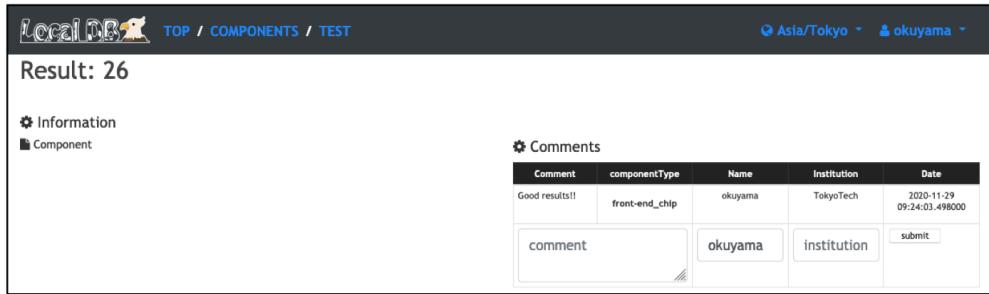


図 4.8 コメント機能

Test Data							
Module Name	Chip Name	Test Type	User	Site	Date	Link	Tag
	JohnDoe_0	std_digitalscan	okuyama	default_host	2020/10/27 15:40:34	result page	anomaly
	JohnDoe_0	std_digitalscan	okuyama	default_host	2020/10/27 15:38:14	result page	good
	JohnDoe_0	std_digitalscan	okuyama	default_host	2020/10/27 15:37:41	result page	anomaly

図 4.9 タグ機能

1 の処理を行う理由は、登録ユーザが関連ソフトウェアを用いて試験結果を MongoDB にアップロードできるようにするためである。2 の情報は、ウェブアプリケーション内のログイン判断、ユーザの情報保持に使う。試験結果アップロードの際にもこの情報を用いて、試験者の記録が行われる。この情報は表 ?? の viewer.user に保存される。2 つの処理について、実際に保存されるドキュメントの例を以下に示す。

Listing 4.1 hoge

```
{
  "_id" : "localdb.hokuyama",
  "userId" : UUID("fee321eb-83b8-434a-a4a0-fff638b5db36"),
  "user" : "hokuyama",
  "db" : "localdb",
  "credentials" : {
    "SCRAM-SHA-1" : {
      "iterationCount" : 10000,
      "salt" : "smkhqvrYnxtsN7rvdadw8g==",
      "storedKey" : "6wC3HyeGTN4px+FV839ou1bylp0==",
      "serverKey" : "/sUu8aX86hnvvMvNq9tC+WZHTDE=="
    },
    "SCRAM-SHA-256" : {
      "iterationCount" : 15000,
      "salt" : "RCC51GuZ4IQK+fINo5HDKP4Vp6LPdersp2gmIA==",
    }
  }
}
```

```
        "storedKey" : "+WwblqTl/SvyyiP7H867kMPPh3Uy2wRIeQ2PgCpdWzs
        "serverKey" : "651T+2BkP1FE4YEHKbxf+JTnzsR9yWcZQPA/y9RN8e0
    }
},
"roles" : [
{
    "role" : "readWrite",
    "db" : "localdb"
},
{
    "role" : "readWrite",
    "db" : "localdbtools"
}
]
}
```

Listing 4.2 hoge

```
{
    "_id" : ObjectId("5f0bbe84ef87af2628865de7"),
    "sys" : {
        "rev" : 0,
        "cts" : ISODate("2020-07-13T10:53:07.943Z"),
        "mts" : ISODate("2020-07-13T10:53:07.943Z")
    },
    "username" : "hokuyama",
    "name" : "Hiroki Okuyama",
    "auth" : "readWrite",
    "institution" : "Tokyo Institute of Technology",
    "Email" : "okuyama@hep.phys.titech.ac.jp",
    "password" : "5f4dcc3b5aa765d61d8327deb882cf99"
}
```

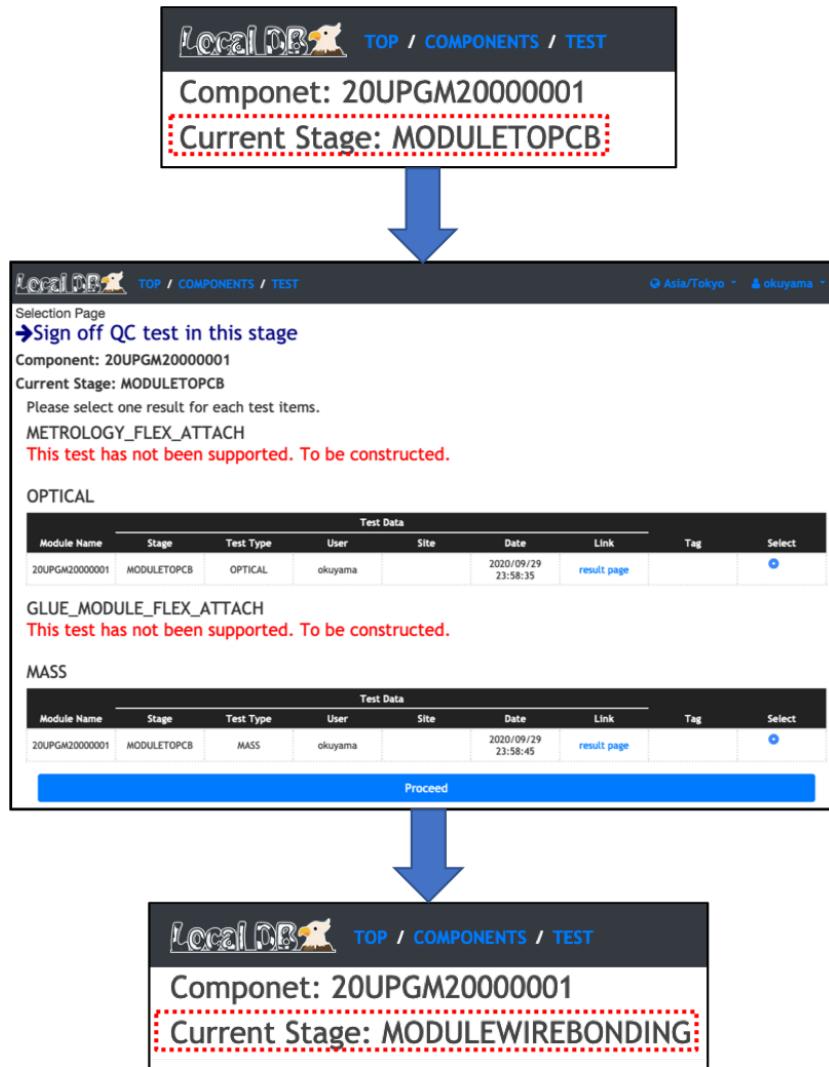


図 4.10 サインオフ機能

4.4.2 品質試験結果の登録と組み立て工程の自動更新

ローカルデータベースへアップロードした品質試験結果の中から、本結果として中央データベースへアップロードする結果を選択する機能を開発した。品質試験は3章で述べたように、各モジュール、各組み立て工程に対して行うものであるため、結果選択も同様に工程毎に行うことを想定している。結果選択後、データベースにおける組み立て工程の情報も次のものへ自動的に更新する機能となっている。

概要

あるモジュール、組み立て工程に対して結果を選択する様子を図??に示す。組み立て工程も自動更新されていることがわかる。

仕組み

Listing 4.3 hoge

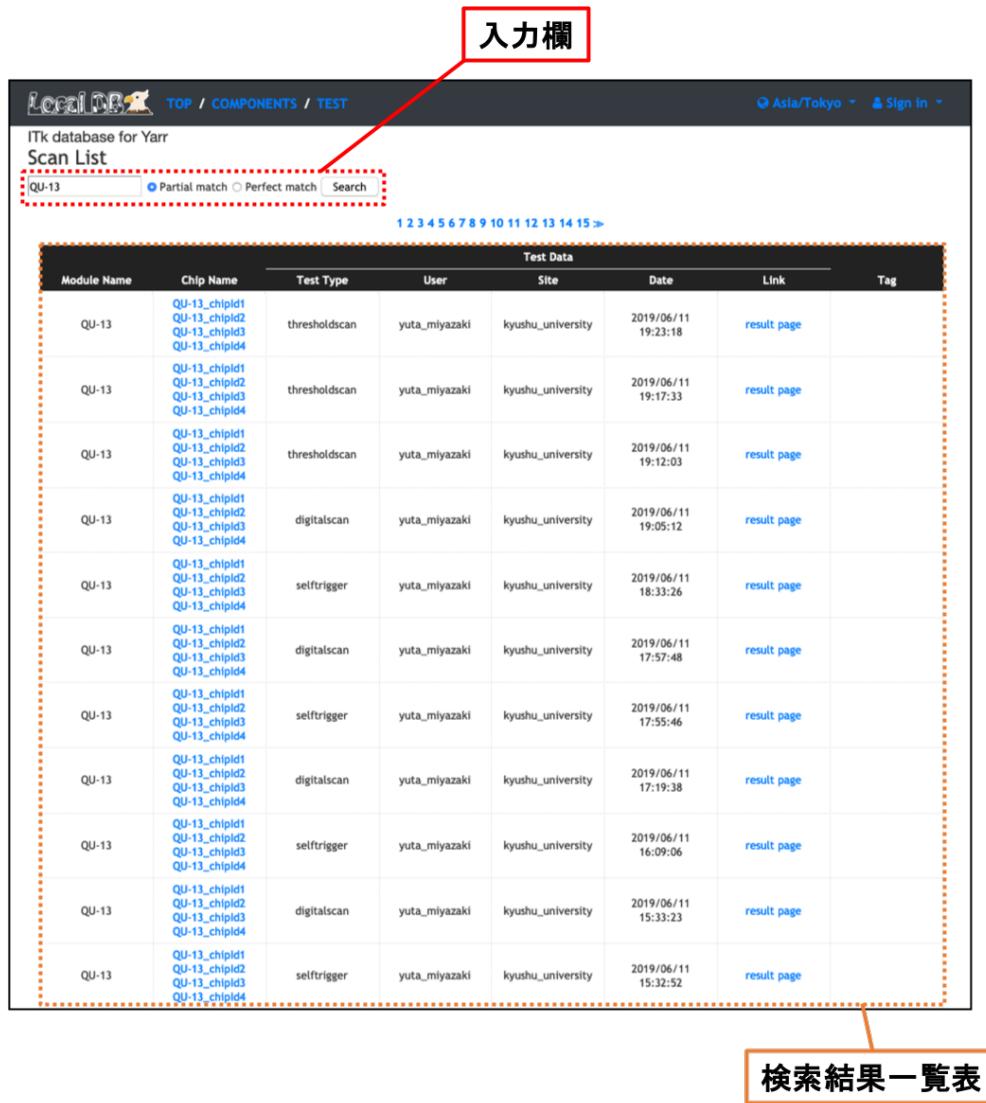
```
{  
    "id" : ObjectId("5fc89aa232d56b29091fd64d"),  
    "sys" : {  
        "mts" : ISODate("2020-12-03T07:58:26.310Z"),  
        "cts" : ISODate("2020-12-03T07:58:26.310Z"),  
        "rev" : 0  
    },  
    "dbVersion" : 1.01,  
    "proddbVersion" : 1.01,  
    "stage_flow" : [  
        "MODULETOPCB",  
        "MODULEWIREBONDING",  
        "MODULEWIREBONDPROTECTION",  
        "MODULEPARYLENECOATING",  
        "MODULETHERMALCYCLING",  
        "MODULEBURNIN",  
        "MODULERECEPTION"  
    ],  
    "stage_test" : {  
        "MODULETOPCB" : [  
            "OPTICAL",  
            "GLUE_MODULE_FLEX_ATTACH",  
            "MASS",  
            "METROLOGY"  
        ],  
        "MODULEWIREBONDING" : [  
            "WIREBONDING",  
            "OPTICAL",  
            "SENSOR_IV",  
            "PIXEL_FAILURE_TEST",  
            "SLDO_VI",  
            "WIREBOND",  
            "CHIP_CONFIGURATION"  
        ],  
        "MODULEWIREBONDPROTECTION" : [  
        ]  
    }  
}
```

```
        "OPTICAL",
        "POTTING",
        "MASS",
        "READOUT_IN_BASIC_ELECTRICAL_TEST",
        "SENSOR_IV",
        "REGISTER_TEST"
    ],
...
},
...
}
```

Listing 4.4 hoge

```
{
    "_id" : ObjectId("5fc4be4c12a45922a91b0e75"),
    "sys" : {
        "mts" : ISODate("2020-11-30T09:41:32.411Z"),
        "cts" : ISODate("2020-11-30T09:41:32.411Z"),
        "rev" : 0
    },
    "dbVersion" : 1.01,
    "proddbVersion" : 1.01,
    "component" : "5fa79114e615fa000a1a5976",
    "currentStage" : "MODULEWIREBONDPROTECTION",
    "latestSyncedStage" : "MODULEWIREBONDING",
    "status" : "created",
    "rework_stage" : [ ],
    "QC_results" : {
        "MODULETOPCB" : {
            "OPTICAL" : "5fc4c2cfb6c93d451e2c9ac1",
            "GLUE_MODULE_FLEX_ATTACH" : "-1",
            "MASS" : "5fc4c2da27766dc6e89c024f",
            "METROLOGY" : "5fc4c2eaf1f19d9cb5859f00"
        },
        "MODULEWIREBONDING" : {
            "WIREBONDING" : "-1",
            "OPTICAL" : "5fc4c4c8b7d0c86912b4958f",
            "SENSOR_IV" : "5fc4c59e9e283a57ccaa1088",
            "PIXEL_FAILURE_TEST" : "5fca342f6e9f1f5eafedb92",
        }
    }
}
```

```
    "SLDO_VI" : "-1",
    "WIREBOND" : "-1",
    "CHIP_CONFIGURATION" : "-1"
  },
  "MODULEWIREBONDPROTECTION" : {
    "OPTICAL" : "-1",
    "POTTING" : "-1",
    "MASS" : "-1",
    "READOUT_IN_BASIC_ELECTRICAL_TEST" : "-1",
    "SENSOR_IV" : "-1",
    "REGISTER_TEST" : "-1"
  },
  ...
}
```



The screenshot shows a web-based application interface for managing test data. At the top, there's a navigation bar with 'LocalDB' logo, 'TOP / COMPONENTS / TEST', and user information 'Asia/Tokyo' and 'Sign In'. Below the navigation is a search bar with a dropdown menu for 'Search type' (set to 'Partial match') and a search button. A red box highlights this search area with the label '入力欄' (Input Box). The main content area is titled 'Scan List' and contains a table titled 'Test Data'. The table has columns: Module Name, Chip Name, Test Type, User, Site, Date, Link, and Tag. The 'Chip Name' column for all rows shows the value 'QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4'. The 'Test Type' column shows various types: 'thresholdscan', 'thresholdscan', 'thresholdscan', 'digitalscan', 'selftrigger', 'digitalscan', 'selftrigger', 'digitalscan', 'selftrigger', 'digitalscan', and 'selftrigger'. The 'User' column shows 'yuta_miyazaki' for all rows. The 'Site' column shows 'kyushu_university' for all rows. The 'Date' column shows dates from '2019/06/11 19:23:18' to '2019/06/11 15:32:52'. The 'Link' column contains blue hyperlinks labeled 'result page' for each row. The entire table is enclosed in a red dashed border. A red arrow points from the 'Search' button in the search bar to the bottom right corner of the table. A red box highlights the bottom right corner of the table with the label '検索結果一覧表' (Search Result List).

Test Data							
Module Name	Chip Name	Test Type	User	Site	Date	Link	Tag
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	thresholdscan	yuta_miyazaki	kyushu_university	2019/06/11 19:23:18	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	thresholdscan	yuta_miyazaki	kyushu_university	2019/06/11 19:17:33	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	thresholdscan	yuta_miyazaki	kyushu_university	2019/06/11 19:12:03	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	digitalscan	yuta_miyazaki	kyushu_university	2019/06/11 19:05:12	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	selftrigger	yuta_miyazaki	kyushu_university	2019/06/11 18:33:26	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	digitalscan	yuta_miyazaki	kyushu_university	2019/06/11 17:57:48	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	selftrigger	yuta_miyazaki	kyushu_university	2019/06/11 17:55:46	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	digitalscan	yuta_miyazaki	kyushu_university	2019/06/11 17:19:38	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	selftrigger	yuta_miyazaki	kyushu_university	2019/06/11 16:09:06	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	digitalscan	yuta_miyazaki	kyushu_university	2019/06/11 15:33:23	result page	
QU-13	QU-13_chipid1 QU-13_chipid2 QU-13_chipid3 QU-13_chipid4	selftrigger	yuta_miyazaki	kyushu_university	2019/06/11 15:32:52	result page	

図 4.11 検索機能の様子

4.4.3 読み出し試験結果の検索機能

登録モジュールや品質試験結果の一覧ページに検索機能を実装した。確認したいモジュール情報や試験結果を迅速に取得し、閲覧できることを目的としている。検索機能を使用している様子を図??に示す。

フリーキーワードを入力し、検索することができる仕組みとなっていて、一般的なウェブページの検索エンジンのように扱うことができる。現在は单一キーワード検索の他に、以下の機能を実装している。

- 完全一致、部分一致検索
- AND、OR 検索

また生産に向けて、検索にかかる処理時間測定を行った。検索機能の詳しい実装方法と処理時間についての詳細は、6章で述べる。

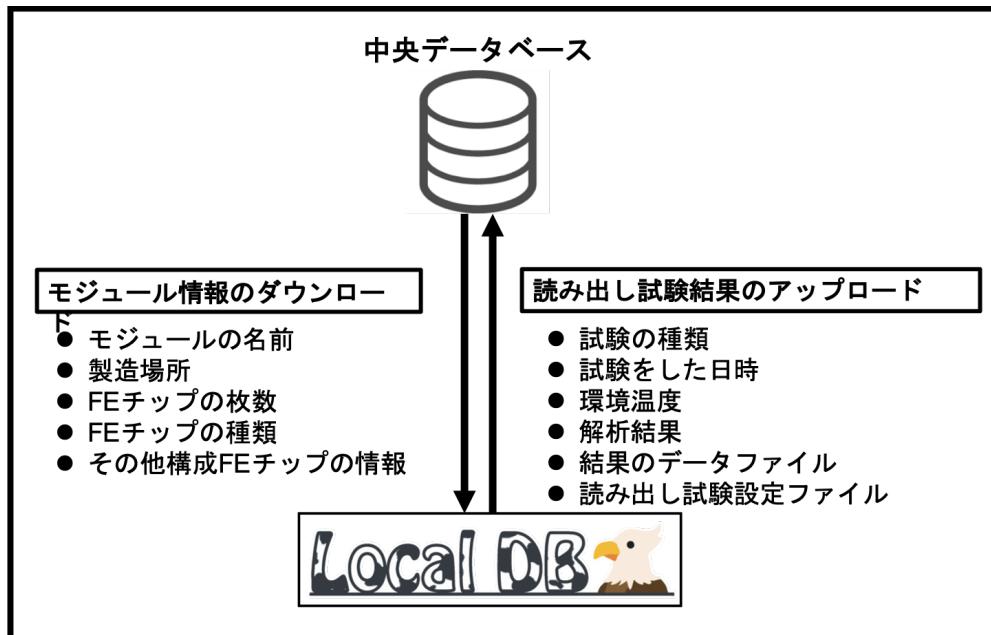


図 4.12 データ同期機能の概要

4.4.4 中央データベースとの情報同期ツール

生産時にはローカルデータベースと中央データベースにおいて、情報の同期が必要となる。例えば、モジュールの ID や組み立て工程、テスト結果といった情報があげられる。この情報同期のためのインターフェースツールを開発した。主に開発した項目については以下の 2 つである。

- モジュール及び構成する FE チップ情報のダウンロード機能
- 読み出し試験結果のアップロード機能

これらの機能のイメージを図??に示す。実装の詳細については 8 章で後述する。

FIGURE

図4.13 データベースシステム操作の流れ

FIGURE

図4.14 生産時のモジュール組み立て状況解析の例

4.5 量産時の情報登録・データ同期の流れ

想定している、モジュールに関しての組み立て工程とデータベースシステムでのユーザ操作の流れを図??に示す。

各ユーザ操作は上述したウェブアプリケーションを用いて行う。最終的には、モジュールの情報及び選択した品質試験の結果が全て中央データベースへ同期されている状態となる。この流れのデモンストレーションを、学内のプロトタイプモジュールを用いて行った。詳細を5章で述べる。

4.6 モジュール生産状況の解析

上述したデータベースシステムを使って、将来的には世界でモジュール生産がどれだけ進んでいるのかを解析する機能を作ることを考えている。全てのモジュールの状況は各生産場所のローカルデータベース上に記録され、組み立て工程ごとに中央データベースへデータ同期する。そのため生産時には、中央データベースで全てのモジュールに関して、現在の組み立て情報を取得できることができ、世界的な生産状況の解析を行うことができる。想定している解析結果のイメージ図を図??に示す。

第5章

不良ピクセル解析ツールの開発と学内実験室における読み出し試験のデモンストレーション

品質試験項目の1つである読み出し試験のデモンストレーションを学内実験室にて行った。デモンストレーションの内容としては、4章で述べた2月にCERNで行ったチュートリアルとほぼ同じ内容である。この章の前半では開発したツールと試験で使用するソフトウェアの概要について説明し、後にデモンストレーションの内容、各ソフトウェアの機能確認について述べる。

5.1 読み出し試験結果解析ツールの開発

品質試験における読み出し試験では、3章で述べたようにモジュールの性能確認のために不良ピクセル解析を行う。これを円滑に行うために、ローカルデータベースシステムに組み込む形でピクセル解析ツールを開発した。このツールについての詳細を以下に示す。

5.1.1 概要

YARRで読み出し試験を行った場合、結果ファイルは digital scan や threshold scan と言ったように各項目ごとにわかれ生成される。また各結果ファイルにはモジュール上の全ピクセル結果が JSON の形で保存されている。結果ファイルを模式的に表したもの以下に示す。

一方、品質試験の不良ピクセル解析においては、いくつかの試験結果を統一的に扱い、かつ各ピクセルごとに解析を行う必要がある。そこで、開発した解析ツールでは複数の結果ファイルを1つのファイルにまとめ、ピクセルごとの解析処理を単純化する役割を担っている。イメージを図??に示す。あるモジュール、組み立て工程における読み出し試験結果の解析を統一的に行うためのツールである。

開発にはCERNが提供している解析フレームワークである ROOT を使用し、C++ を用いた。いくつかの試験の統一ファイルとして、ROOT内部の機能である Tree を使用した。

実際に作った Tree ファイルと、このファイルのデータ保持のイメージをそれぞれ図??、??に示す。

5.1.2 ツールの内部構造と処理の流れ

開発したツールは、主に以下で説明する3つの実行ファイルで構成される。それぞれの役割について説明する。

getDataFile.py (Python)

データベースから対象となるデータファイルを取得、キャッシュファイルとしてサーバー上の一時ディレクトリに保存

makeTree (C++)

getDataFile.py を用いて生成されたキャッシュファイルを読み込み、Tree ファイルを作成

analysis (C++)

作成した Tree ファイルを読み込み不良ピクセル解析を実行、結果値やプロットを出力

処理の流れのイメージを図??に示す。

5.2 読み出し試験に用いるソフトウェアの概要

試験で用いるソフトウェアをいかに示す。

- YARR
- MongoDB
- ウェブアプリケーション
- 中央データベースとのデータ同期ツール
- ピクセル解析ツール
- InfluxDB
- Grafana
- 電源操作用ソフト
- 温度読み出し用ソフト

5.3 学内実験室におけるデモンストレーション

学内実験室で開発しているソフトウェアを用いて読み出し試験を行い、実際の生産時における流れのデモンストレーションを局所的に行なった。その詳細について以下に示す。

5.3.1 デモンストレーションの流れ

今回のデモンストレーションで確認した機能を以下に示す。

- 中央データベースとローカルデータベースのデータ同期機能 (モジュール ID のダウンロード、試験結果のアップロード)
- 読み出し試験に使う各種機能 (設定ファイル生成、温度、電圧、電流モニタリング、試験結果閲覧)
- 結果選択とピクセル解析機能

FIGURE

図 5.1 デモンストレーションの流れ

表 5.1 各ハードウェアの性能

1	2
result 1	result 2

FIGURE

図 5.2 ハードウェアセットアップの概要

FIGURE

図 5.3 各ハードウェアの写真

FIGURE

図 5.4 ダウンロードしたモジュール ID 確認画面

またデモンストレーションにおける流れの概要を図??に示す。

5.3.2 読み出し試験セットアップ

読み出し試験に用いるハードウェアのセットアップを表??、概要を図??、各ハードウェアの写真を??に示す。各装置の詳細については付録 B に示す。

5.3.3 読み出し試験内容

読み出し試験を通して、モジュールに与える電圧値、電流値、チップ横についている NTC から読み取れる温度を記録した。以下の流れに沿って読み出しを行なった。

5.3.4 機能確認

モジュール ID のダウンロード

登録したモジュールの ID を機能を使ってダウンロードし、ウェブアプリケーションで確認した。確認した画面を図??に示す。

読み出し試験

以下の流れで読み出し試験を行なった。読み出し試験はサーバーのシェルを用いて行う。

- ・設定ファイル生成

ダウンロードしたモジュールの ID を用いて、読み出しに用いる設定ファイルを生成した。イメージを図??、実際に生成したファイルを確認した画面を図??に示す。

- ・試験実施とアップロード

上述した流れに沿って読み出し試験を実施した。試験結果は各試験の終わりに自動的にアップロードさ

FIGURE

図 5.5 設定ファイル生成のイメージ

FIGURE

図 5.6 生成ファイル確認画面

FIGURE

図 5.7 DCS のモニタリング

FIGURE

図 5.8 試験結果の閲覧

FIGURE

図 5.9 各測定値の閲覧

FIGURE

図 5.10 scan 結果選択の様子

表 5.2 ピクセル解析結果

1	2
result 1	result 2

表 5.3 scan file の存在確認

1	2
result 1	result 2

れるようなシステムとなっている。

- ・電圧値、電流値、温度のモニタリング

記録した値を Grafana を使ってモニタリングをした。その様子を図??に示す。

- ・検索機能の確認検索機能の確認を行った。
- ・試験結果の閲覧

ウェブアプリケーションを用いて、試験結果を閲覧した。その様子を図??、??に示す。

結果選択とピクセル解析

読み出し結果を選択し、ピクセル解析を行なった。結果選択の様子を図??、解析結果を表??に示す。

試験結果アップロード

選択した結果を中央データベースにアップロードし、各ファイルが正しくアップロードされていることを確認した。各ファイルの存在を確認した結果を表??に示す。

第 6 章

ローカルデータベースにおける読み出し 試験結果検索機能の詳細と処理時間測定

生産時には、読み出し試験の結果は一つの機関で大量に生じるものである。4章で述べたように、任意のタイミングで必要な結果を取得できる検索機能を実装した。詳細について以下に示す。

6.1 実装方法

今回の実装では、一般的にウェブで用いられているフリーワードの検索エンジンのような機能を実装しようと考えた。ユーザの操作を最小限にし、柔軟な検索ができるようにするためにである。

読み出し試験において、対象とする検索キーワードを以下の項目に絞った。システムにおいて、アップロードされた試験結果に関するデータベース内の情報は固定されていて、試験結果情報の変更はしないことを前提としている。そのため、ユーザが対象としたい検索キーワードは以下の項目に限られ、検索キーワードとして以下の項目をサポートすれば十分であると考えた。

- モジュール及び FE チップの ID
- 読み出し試験項目 (例:std_digitalscan)
- 読み出し試験者
- 読み出し試験場所
- 試験日時 (将来的に範囲指定を用いた検索機能を検討)
- タグ機能を用いてつけられたタグ

そこで実装方法として、以下の 2 つを考えた。

1. 各試験に関する情報を Python リスト集め、検索キーワードが含まれるかを確認する方法
2. 各試験に関する情報を持つドキュメント、コレクションを予め作成、それを参照し検索を行う方法

これらについて以下で詳細を説明する。

6.1.1 方法 1: Python リストを用いた一致確認

Python リストを使う実装の場合、以下のような流れで検索処理を行う。

1. ユーザが検索ワードを入力し、処理を実行

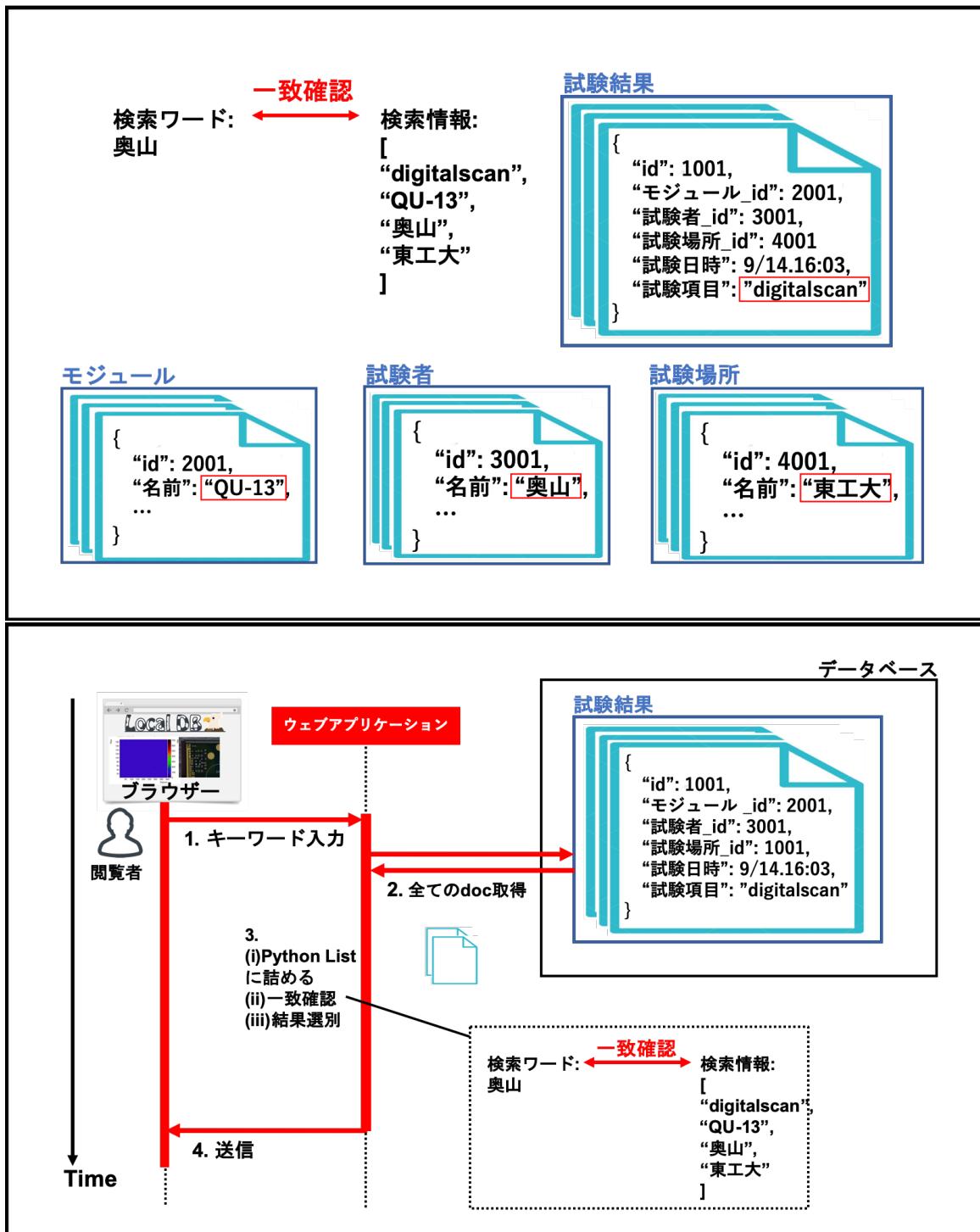


図 6.1 検索機能実装方法 1:Python リストを用いた場合

- 読み出し試験に関する情報を全て取得
- Python リストに保持、検索ワード一致を確認、試験を選別
- ブラウザーに送信

アルゴリズムのイメージを図??に示す。この方法では、データベース内の試験結果とアプリケーションの関数内だけで全ての処理を行うことが可能なため、シンプルな実装方法である。

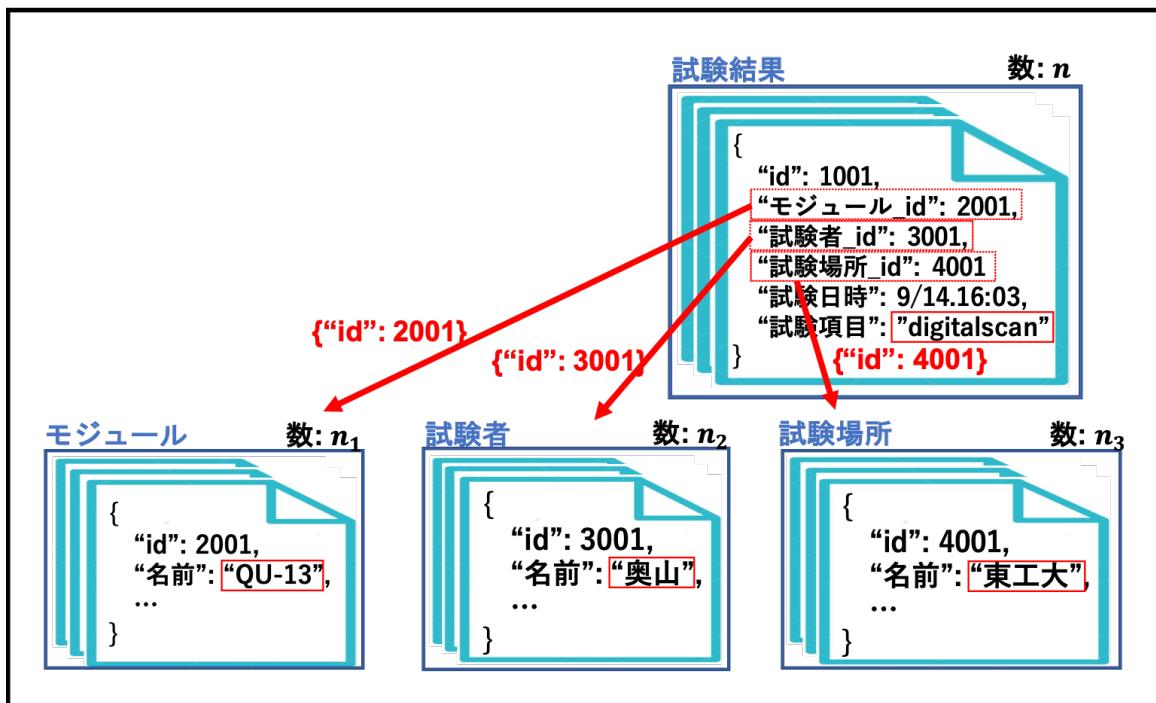


図 6.2 検索機能実装方法 1 の問題点

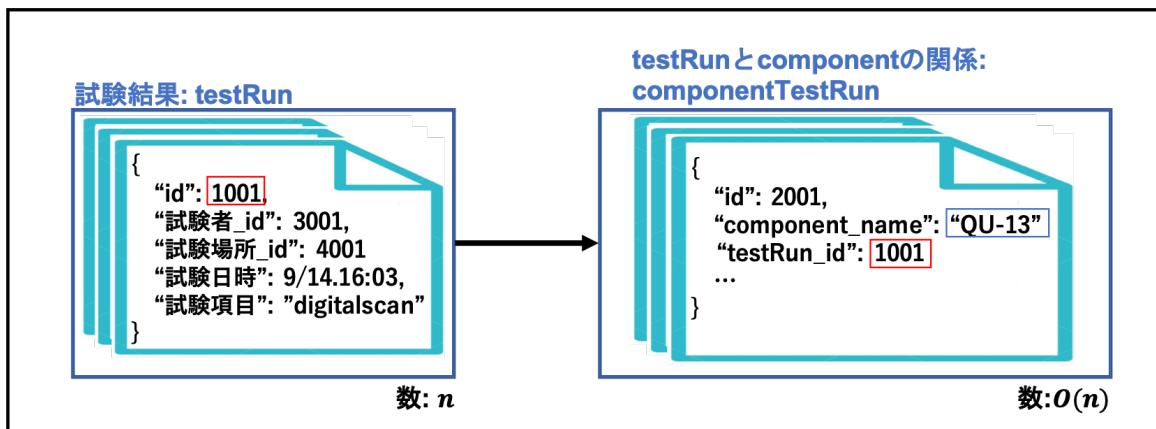


図 6.3 検索機能実装方法 1 の問題点詳細

ユーザが処理を実行した際にデータベース内で情報を取得し、Python リストにつめる処理を行う。

しかしこの方法を試験実装したところ、データベース内の構造は複雑であり複数のコレクションを跨いで情報を保持しているため、試験結果全てに対してリアルタイムでこの処理を行うと、時間を大きく要してしまう問題が発生した。イメージを図??に示す。

このシステムにおいては試験結果数を n とすると、試験結果情報を保存している testRun、component と testRun の関係を保存する componentTestRun がそれぞれ $O(n)$ のドキュメント数を持つことになる。各 component の情報を取得し、一致確認を行うと $O(n^2)$ の時間がかかる。イメージを図??に示す。

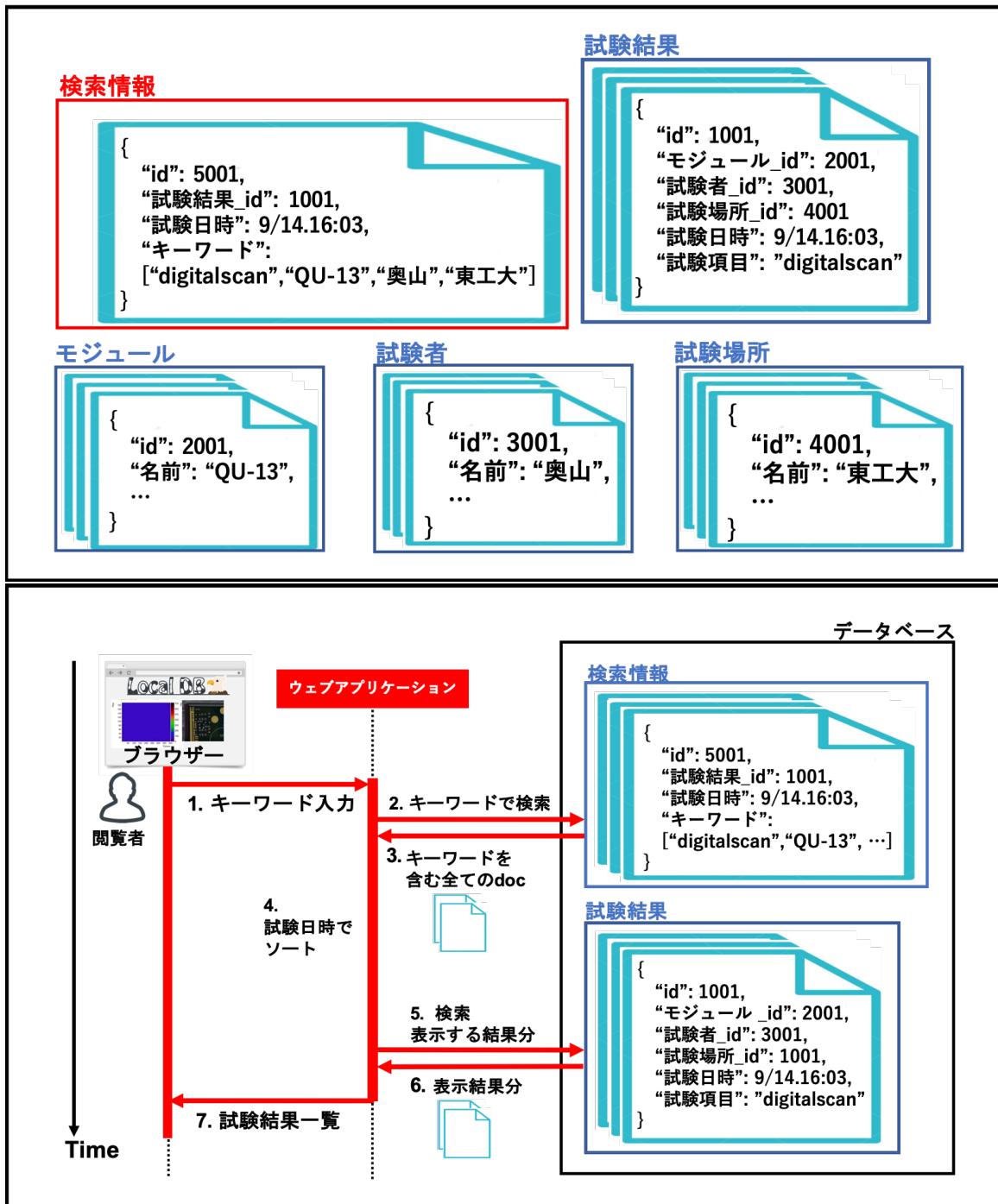


図 6.4 検索機能実装方法 2:検索キーワード専用コレクションを用いた場合

6.1.2 方法 2: 検索情報を持つドキュメントを作成、使用

検索機能を改善するため方法 2 を考案し、実装を行った。アルゴリズムのイメージを図??に示す。検索キーワードを別のドキュメントに予め保持しておき、処理実行時にはそれを参照することで検索を行う。検索情報コレクションに入るドキュメント数は、試験結果数と同じであるためドキュメント数に対する処理時間は $O(n)$ と考えられ、検索コストを削減できると思い、この方法を考案した。

以下に検索情報のドキュメントの例を示す。

表 6.1 ノート PC の性能

PC名	CPU	Type	Core	Thread	Clock speed[GHz]	Memory[GB]	Disk[GB]
MacBookAir(13-inch,2017)	Intel Core i5	2	4	1.8		8	256

Listing 6.1 hoge

```
{
    "_id" : ObjectId("5fd489f60e2ca70557e44a8b"),
    "runId" : "5fc4d027b1ef7c6297c91040",
    "timeStamp" : ISODate("2020-11-30T10:57:19Z"),
    "data" : [
        "20UPGR00000001",
        "20UPGFC9999999",
        "std_digitalscan",
        "hokuyama",
        "tokyo_institute_of_technology",
        "2020/12/09"
    ]
}
```

6.2 処理時間測定

先述した方法1と2において、検索処理時間の測定を行った。詳細を以下に示す。

使用したデバイス

測定には個人で使用しているノートPC(MacBookAir(13-inch,2017))を用いた。性能を表??に示す。

測定内容

コマンドプロンプトから以下を使用し、リクエストに対するアプリケーションのレスポンス時間を測定した。実際にアプリケーションを使用する際には、ブラウザに一覧表示をする時間が今回の測定時間に加算されることになる。

```
curl "http://127.0.0.1:5000/localdb/scan?keywords=okuyama&match=partial"
-o /dev/null -w "%{time\_total}\n" 2> /dev/null -s
```

上記のコマンドを用いて、試験結果数に対するレスポンス時間の測定を行った。測定に関する詳細を表??に示す。

各測定点に対して平均値、標準偏差を算出し、フィッティングを行った。

表 6.2 検索機能処理時間測定の詳細

試験結果数	1, 5000, 10000, 15000, 20000
測定回数	各測定点に対して 20 回
検索キーワード	okuyama
検索モード	部分一致
各試験結果が持つ検索情報	全試験結果に対して同じ、以下に詳細
検索情報一覧	モジュール名: 20UPGR10000005 FE チップ名: 20UPGTU9000000 試験項目: std_digitalscan 試験者: okuyama 試験場所: default_host 試験日時: 2020/12/06

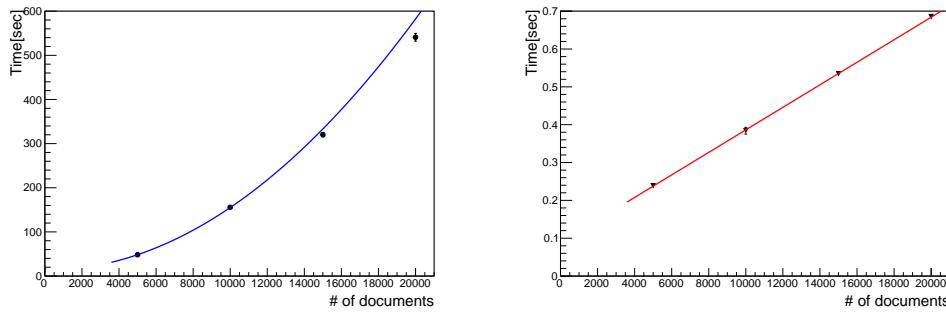


図 6.5 検索処理速度測定

測定結果

結果を図??に示す。また得られた近似関数をそれぞれ式??、??に示す。方法 1 に対して、方法 2 の処理時間は改善されていることが分かる。

$$y = \{(1.4 \pm 0.0) \times 10^{-6}\}x^2 + (13 \pm 0) \quad (6.1)$$

$$y = \{(3.0 \pm 0.1) \times 10^{-5}\}x + \{(8.9 \pm 0.8) \times 10^{-2}\} \quad (6.2)$$

現在は方法 2 で検索機能を実装し、サービスの 1 つとして提供している。

生産時における検索時間の見積もり

各方法について、生産時における処理時間の見積もりを行う。簡単のため今回使用したデバイスと生産時に使うサーバーの性能差は無視する。ここでデータベースで管理するモジュール数は日本が最多とし、その数を予定している 2,000 とする。保存する読み出し試験数は 3 章で述べたように、1 つのモジュールあたり 42 とする。全ての生産が終了した際の検索処理時間を見積もる。上で得られた関係式を用いて検索処理実行時間は方法 1、2 に対して式??、??のように見積もることができる。

$$\{(1.4 \pm 0.0) \times 10^{-6}\} \times (2000 \times 42)^2 + (13 \pm 0) = (9.8 \pm 0) \times 10^3 [\text{sec}] \quad (6.3)$$

$$\{(3.0 \pm 0.1) \times 10^{-5}\} \times (2000 \times 42) + \{(8.9 \pm 0.8) \times 10^{-2}\} = 2.6 \pm 0.1[\text{sec}] \quad (6.4)$$

方法1では約9800秒の処理時間が見積もられ、生産時には何台ものモジュール結果の保存、管理をすることから運用不可能である。方法2では終了時点においても数秒で処理を終えることができるため、生産を通して十分に運用可能であると考えれる。

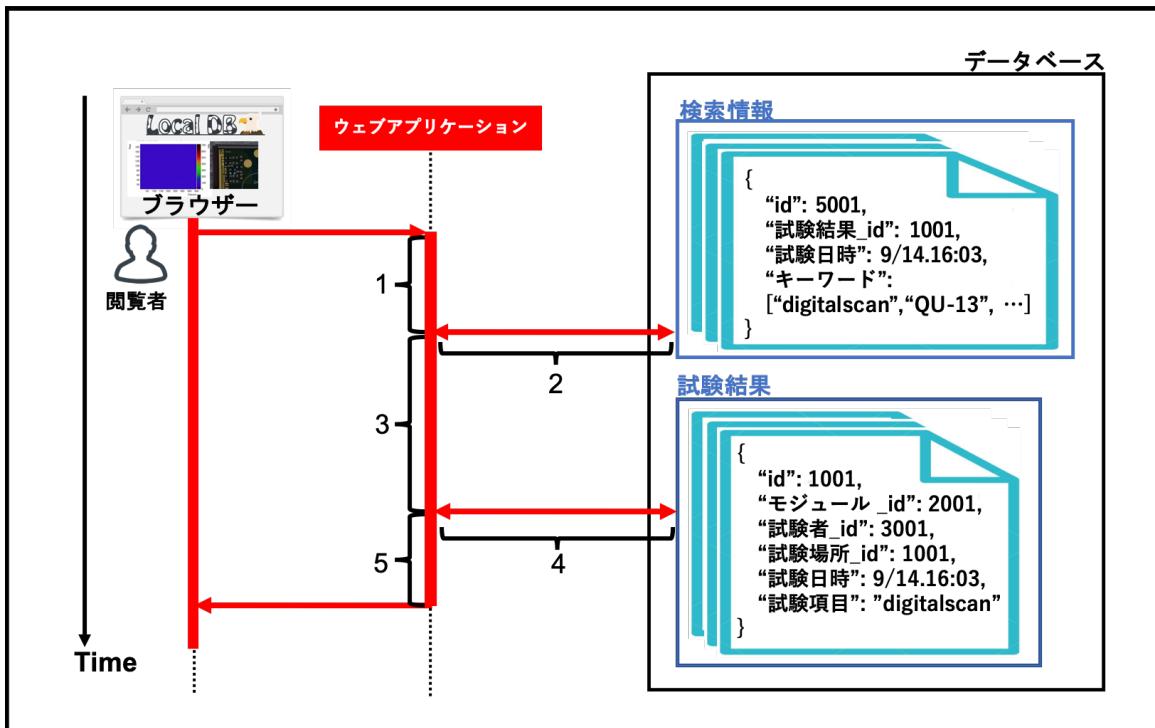


図 6.6 方法 2 における検索処理詳細

6.3 処理時間の改善方法の考案と実装

更なる処理時間の改善を目的として、新たな処理アルゴリズムの考案と測定を行った。詳細について以下に示す。

6.3.1 方法 2 における検索処理時間の内訳

先述したように、方法 2 では処理時間が改善した。この方法 2 について、処理時間の内訳を知るために追加で測定を行った。アプリケーション層での各処理について、以下のように番号を付ける。

1. キーワードを受け取り、検索情報コレクションに検索をかけるまでの処理
2. 検索情報コレクションに検索をかけ、情報を受け取る処理
3. ドキュメントを受け取り該当する試験結果の Id をまとめ、検索をかけるまでの処理
4. 試験結果コレクションに検索をかけ、情報を受け取る処理
5. ドキュメントを受け取りデータを整形、ブラウザにレスポンスを返すまでの処理

イメージを図??に示す。

ボトルネックとなっている処理を測定するために、各処理にかかる時間の測定を行った。測定に関しての詳細を表??に示す。

結果を図??に示す。

処理 3,5 の割合が大きいことがわかる。これらについてさらに内訳を調べたところ、以下の処理の割合

表 6.3 検索機能処理時間測定の詳細

試験結果数	10000
測定回数	20 回

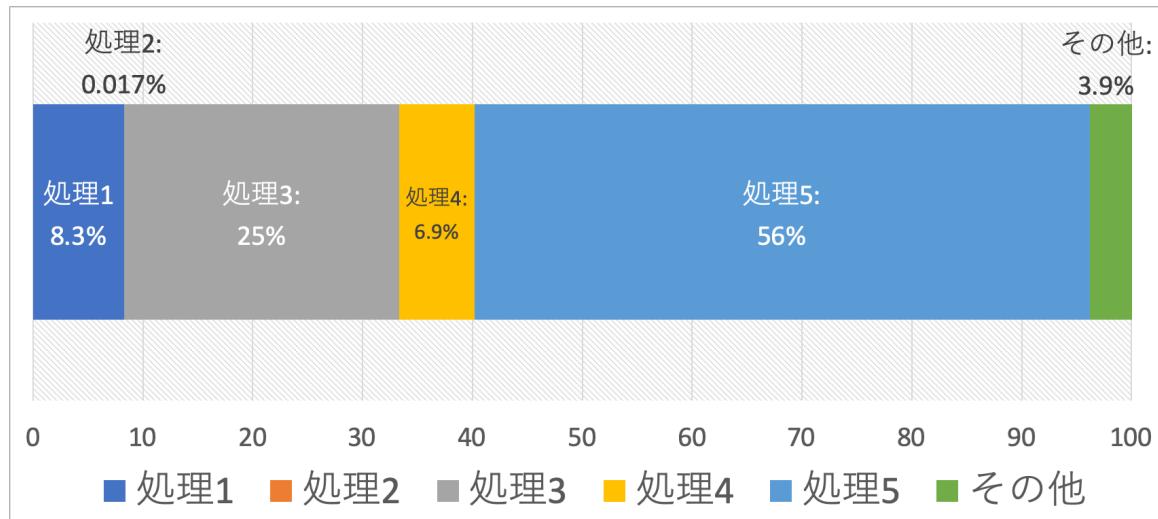


図 6.7 方法 2 における検索処理詳細結果

表 6.4 処理 3,5 における処理??の割合

処理	全体 [sec]	処理??[sec]	割合 [%]
3	0.091	0.089	97
5	0.21	0.18	86

が大きいことがわかった。

取得した複数ドキュメントを Python リストに変換する処理

(6.5)

上述した方法 2 における処理 3,5 において、処理??の割合を表??まとめた

またある検索時におけるコレクション内部の全ドキュメント数と処理??に要する時間の関係を図??に示す。全ドキュメント数に対して線形性を示していることがわかる。

6.3.2 改善点

上記の測定を踏まえ、改善方法として以下の項目を検討した。

- 検索処理の回数を減らす
- 検索対象コレクションのドキュメント数を減らす

上述した 2 つを目的として、以下の 2 つの方法を新しく考え処理時間測定を行った。

3. 検索情報のコレクションに一覧表示に必要な情報を保持、参照
4. 方法 3 に付け加えて、検索情報のドキュメントを複数コレクションに分散、マルチスレッド

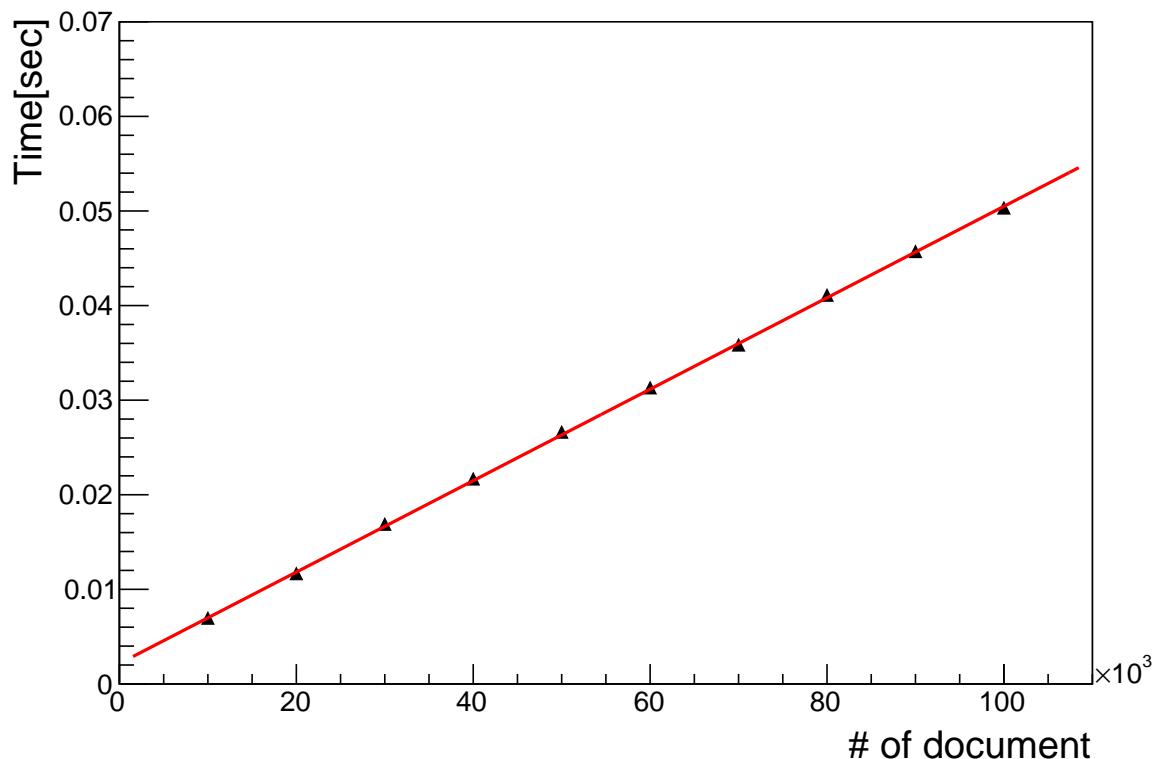


図 6.8 ドキュメント数に対するリスト変換処理時間の関係

方法 3 については処理の数を減らし、データベースに対する検索の回数を減らすことを目的としている。方法 4 については処理の数を減らすことに加えて、ドキュメントの数を分散し並列処理することで処理時間の改善を図っている。イメージをそれぞれ図??、??に示す。

方法 3,4 について、章??と同じ内容の測定を行った。方法 2 のものと合わせた結果を??に示す。処理時間の改善に成功した。得られた方法 3,4 に関する関係を式??、??に示す。

$$y = \{(2.9 \pm 0.1) \times 10^{-5}\}x + \{(5.0 \pm 11) \times 10^{-3}\} \quad (6.6)$$

$$y = \{(2.7 \pm 0.1) \times 10^{-5}\}x + \{(2.2 \pm 1.0) \times 10^{-2}\} \quad (6.7)$$

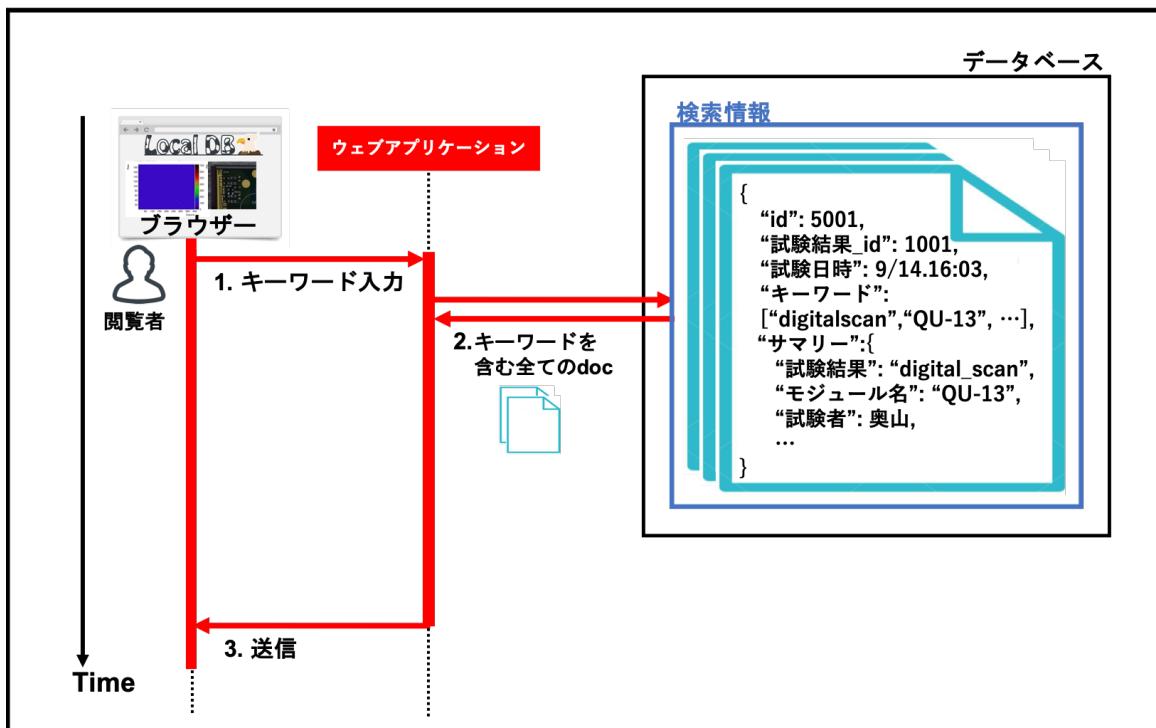


図 6.9 検索機能実装方法 3:検索情報と共に一覧表示に必要な情報を保持

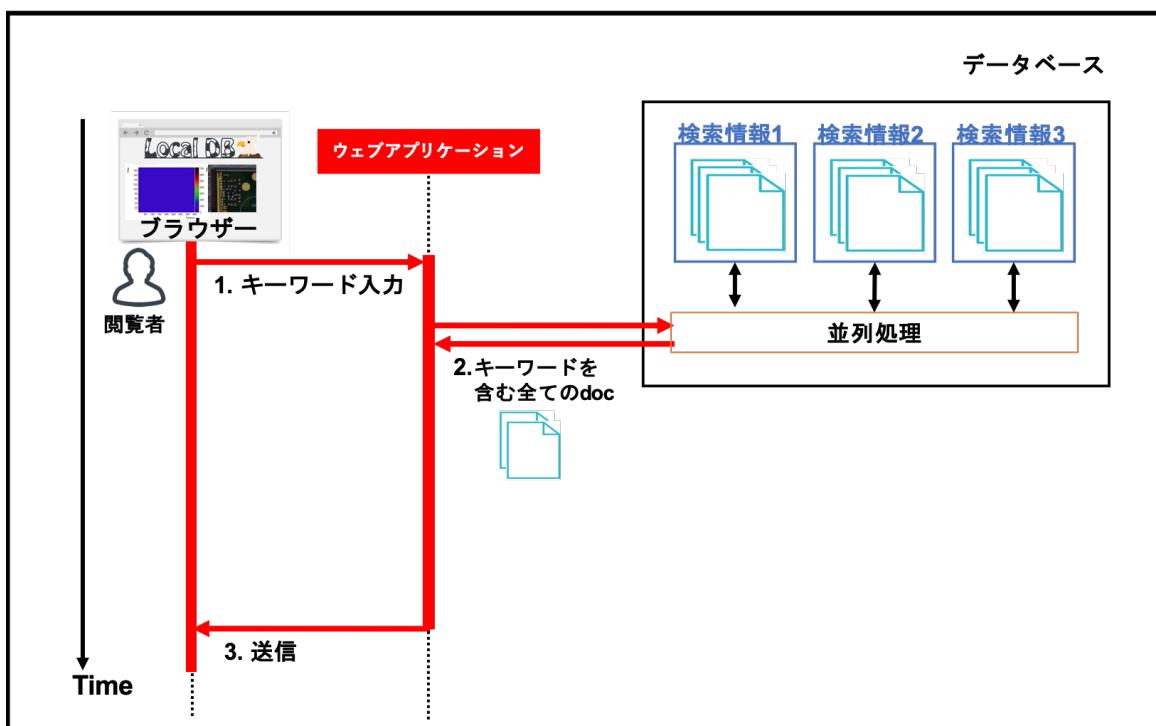


図 6.10 検索機能実装方法 4:検索情報コレクションを分散、マルチスレッドを用いる

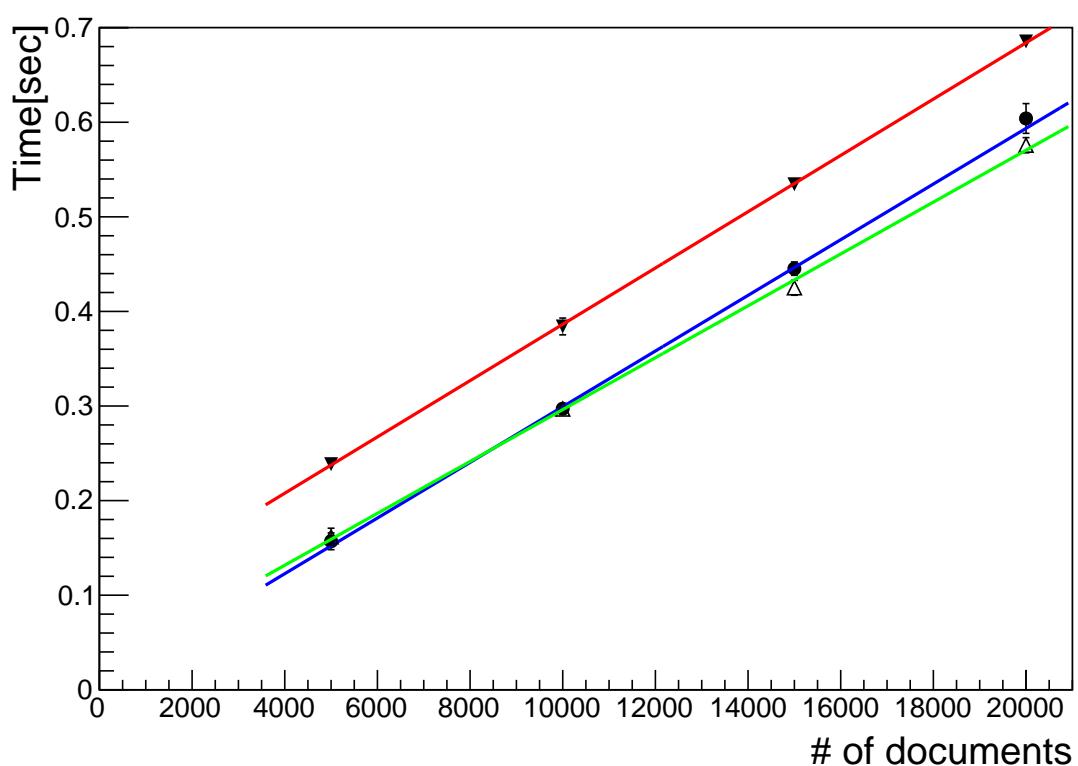


図 6.11 検索機能実装方法 3,4 に対する測定

第7章

中央データベースとローカルデータベースのデータ同期ツールに関する開発

開発した機能が生産時に十分であるかどうか見積もりは今後の開発を効率よく進めていく上で重要である。生産時のデータの数や量を想定してその際のシステム性能を見積もることで、今の実装で十分かどうか、改善が必要な場合どのように改善すればいいかを知ることができる。中央データベースとローカルデータベースのデータ同期機能に関して処理時間評価を行った。詳細について以下で説明する。

7.1 サーバーの設置場所による処理時間の違い

4章で述べたように、中央データベースはチェコに設置されている。そのため試験結果のアップロードに関して、各組み立て機関から接続しデータ送信する処理時間は、機関の場所に大きく依存すると考えられる。世界的にデータ同期ツールが不自由なく動くことを向けて開発、改善に役立てることを目的として、データを送信する処理時間を、以下の3つの場所に置かれているサーバーを用いて測定した。

- 日本、高エネルギー加速器研究所 (KEK)
- アメリカ、バークレー研究所 (LBL)
- スイス、欧州原子核研究機構 (CERN)

各サーバーの性能を表??に示す。また各サーバーが置かれている場所の位置関係を図??に示す。

これらのサーバーは実際に生産の際に使用するものと同程度の性能を持ち、サーバーが置かれている環境も生産時と同じであるとしている。回線の混雑具合などによる処理時間の低下は、本測定では考慮に入れていない。

表 7.1 サーバーの性能一覧

設置機関	CPU Type				Memory [kB]	Disk [GB]
		Core	Thread	Clock speed[GHz]		
KEK	Intel(R) Core(TM) i7-9700K	8	16	3.6	32,658,772	1800 + 1800
LBL	Intel(R) Core(TM) i7-8700	6	12	3.7	32,628,000	233
CERN	Intel(R) Core(TM) i7-4790	4	8	3.6	32,686,404	238.5 + 3700 + 3700



図 7.1 各サーバの位置関係

表 7.2 データ同期ツールの中で使用した API

関数名	処理の内容	本ツールでの使用用途
getComponent	登録した部品情報の取得	主にダウンロード時におけるモジュールやチップの情報取得に用いる。
listComponents	登録した部品情報一覧の取得	主にダウンロード時におけるモジュール情報一覧取得に用いる。
uploadTestRunResults	テスト結果生成	読み出し試験結果生成の際に用いる。
createTestRunAttachment	ある試験結果に対するバイナリファイルの添付	読み出し試験結果生成後にファイルを添付する際に用いる。

7.1.1 データ同期ツールに使用する API

中央データベースのデータ取得には、用意されている API のサービスを使用している。ローカルデータベースとのデータ同期ツールの中で主に使用している API を表??に示す。

7.1.2 API 使用にかかる時間

上述した API 使用時の処理時間を各サーバーで測定した。以下の 3 つの測定を行なった。

- getComponent を用いた、登録モジュール情報 1 つの取得時間測定
- createTestRunAttachment を用いて、ある試験結果ページに 1Byte のデータファイルを添付する時間測定
- createTestRunAttachment を用いて、ある試験結果ページに容量の異なるデータファイルを添付、容量に対する時間依存性を測定

最初の 2 項目に関して、各処理時間についてまとめたものを表??、??に示す。またファイル容量と処理時間の関係を図??に示す。1Byte での測定点も含んでいる。

7.2 モジュール ID のダウンロード機能確認と処理時間測定

7.2.1 ダウンロードする情報と構造

中央データベースから、モジュール及び FE チップの情報をダウンロードする。それぞれの情報の詳細について表??に示す。

表 7.3 モジュール情報の取得にかかる時間

サーバー	処理時間 [秒]
KEK	0.49 ± 0.02
LBL	0.37 ± 0.02
CERN	0.30 ± 0.04

表 7.4 1Byte のデータファイル添付にかかる処理時間

サーバー	処理時間 [秒]
KEK	0.54 ± 0.04
LBL	0.34 ± 0.03
CERN	0.39 ± 0.02

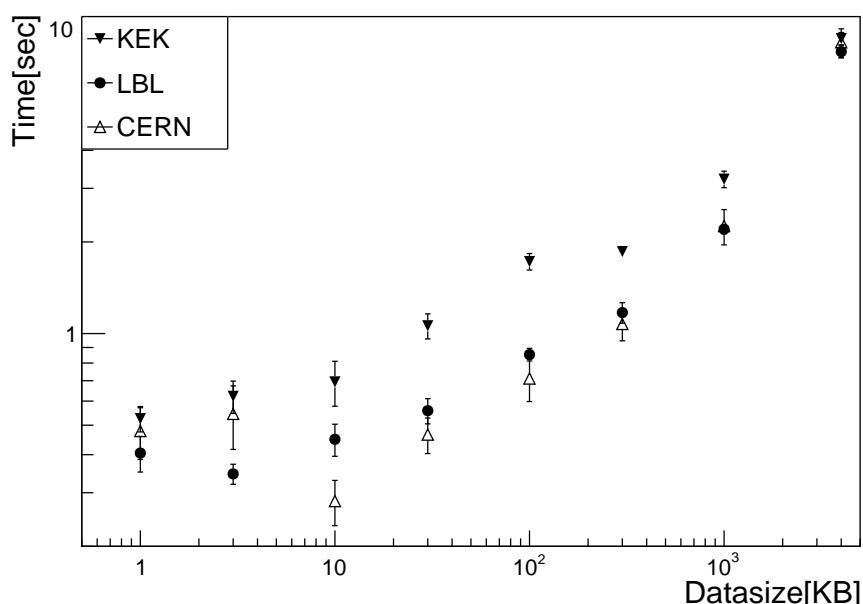


図 7.2 添付するファイルサイズと処理時間の関係

表 7.5 ダウンロードする情報

部品	情報
モジュール	シリアルナンバー
	搭載 FE チップの種類
	登録機関
	搭載 FE チップの枚数
FE チップ	シリアルナンバー
	FE チップ ID(モジュール上の位置を表す情報)
	登録機関

ダウンロードされたモジュール、FE チップのドキュメントの例を以下に示す。

Listing 7.1 モジュール

```
{  
    "_id" : ObjectId("5fa79114e615fa000a1a5976") ,  
    "name" : "20UPGR00000001" ,  
    "chipType" : "RD53A" ,  
    "serialNumber" : "20UPGR00000001" ,  
    "chipId" : -1,  
    "componentType" : "module" ,  
    "address" : "5fd597fdf7339bbf26b87fb2" ,  
    "children" : 1 ,  
    "sys" : {  
        "mts" : ISODate("2020-12-13T04:26:37.989Z") ,  
        "cts" : ISODate("2020-12-13T04:26:37.989Z") ,  
        "rev" : 0  
    } ,  
    "dbVersion" : 1.01 ,  
    "user_id" : -1 ,  
    "proDB" : true  
}
```

Listing 7.2 FE チップ

```
{  
    "_id" : ObjectId("5fa79560e615fa000a1a5a16") ,  
    "name" : "20UPGFC9999999" ,  
    "chipType" : "RD53A" ,  
    "serialNumber" : "20UPGFC9999999" ,  
    "chipId" : 0 ,  
    "componentType" : "front-end-chip" ,  
    "address" : "5fd597fdf7339bbf26b87fb2" ,  
    "children" : -1 ,  
    "sys" : {  
        "mts" : ISODate("2020-12-13T04:26:37.984Z") ,  
        "cts" : ISODate("2020-12-13T04:26:37.984Z") ,  
        "rev" : 0  
    } ,  
    "dbVersion" : 1.01 ,  
    "user_id" : -1 ,
```

表 7.6 ダウンロード処理時間測定

モジュール	処理時間
20UPGM20030004	3.8
20UPGM20030001	3.7
20UPGM20030003	5.9
20UPGM20030006	3.6
20UPGM20030022	3.8
20UPGM20030024	3.3
平均	4.0 ± 0.4

```

    "proDB" : true
}

```

7.2.2 処理の流れ

ダウンロード機能における処理の流れのイメージを図??に示す。

7.2.3 機能確認

KEK で組み立てられた 6 台のモジュールを中央データベースに登録し、ダウンロードを行った。登録したモジュールを表??、ダウンロードをしてアプリケーションで確認した様子を図??に示す。

7.2.4 処理時間測定

ダウンロードした際の処理時間を測定した。これについてまとめたものを表??に示す。

7.2.5 生産時における見積もり

現在ダウンロード機能のオプションとして、以下の 2 つを実装している。

1. モジュール 1 つのシリアルナンバーを打ち込み、そのモジュールをダウンロードする機能
2. 登録されている全てのモジュールを一括でダウンロードする機能

オプション 1 の見積もり値は、表??の平均値として $4.0 \pm 0.4[\text{sec}]$ となる。オプション 2 の見積もり値は、生産時には最大で 10,000 台のモジュールを生産することから、以下のようになる。

$$(4.0 \pm 0.4) \times 10,000 = (4.0 \pm 0.4) \times 10^4[\text{sec}] \quad (7.1)$$

$$= 11 \pm 1[\text{hour}] \quad (7.2)$$

このダウンロード機能についてはモジュール登録機能の直後に使用する設計となっている。モジュール登録は各機関が各モジュールの組み立てを始める時にやることを想定していて、これは 1 つずつ行われるため、現在はオプション 1 のみを提供している。

しかし、将来的には世界中で様々なモジュール組み立ての流れが想定される。例えば機関 1 で何台かのモジュールを登録した後に、機関 2 に輸送するような場合、機関 2 では機関 1 で登録されたモジュールを

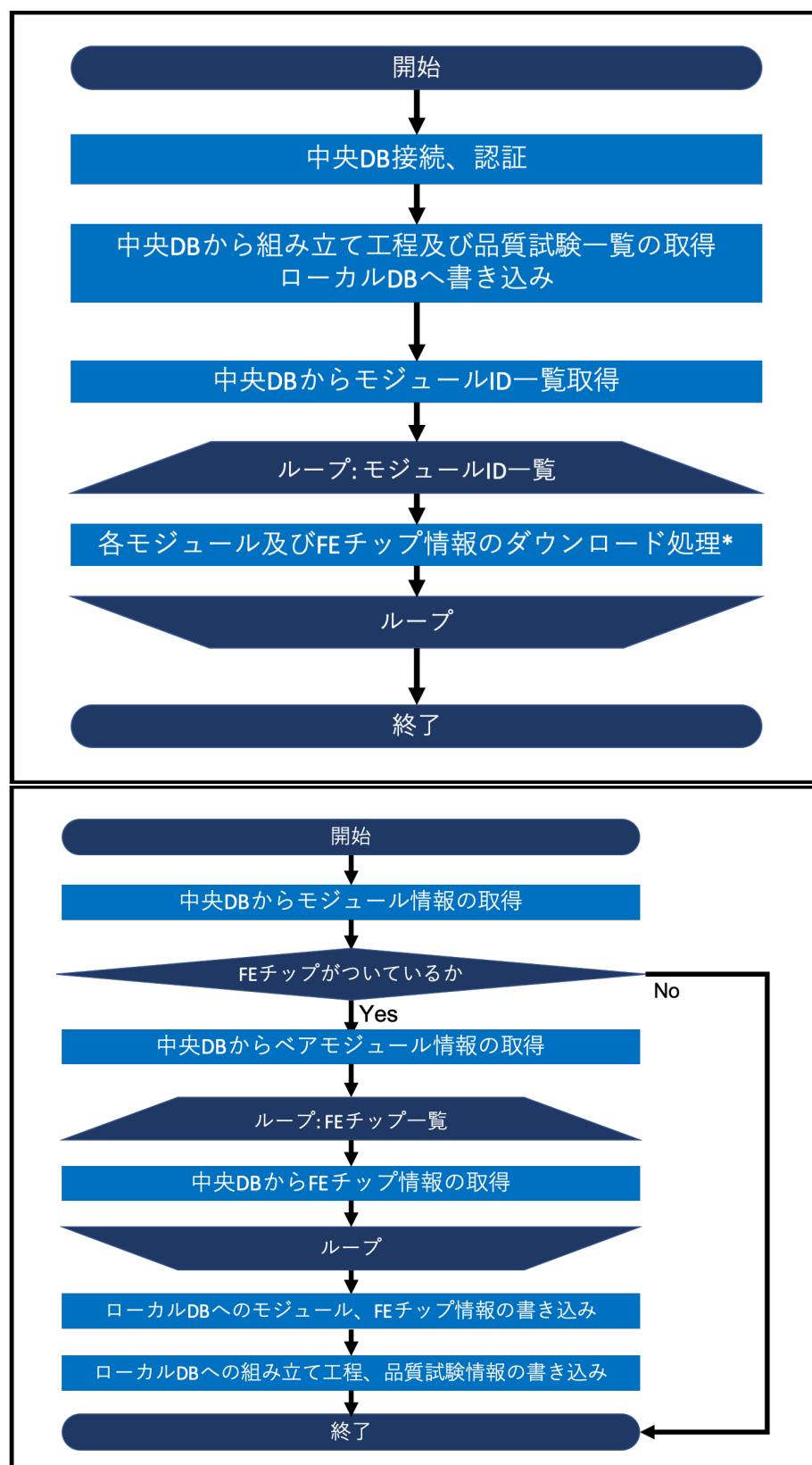


図 7.3 ダウンロード処理に関する流れのイメージ図

Module	Bare Module	Sensor	PCB	Carrier	FE chip
20UPGM20030004	20UPGB40500019	20UPGS83300002	20UPGPQ0030004	20UPGMC0210000	20UPGFC0014659 20UPGFC0014658 20UPGFC0014675 20UPGFC0014691 20UPGFC0014644 20UPGFC0014628 20UPGFC0014660 20UPGFC0014708 20UPGFC0014677 20UPGFC0014629 20UPGFC0014693 20UPGFC0014646 20UPGFC0016282 20UPGFC0016281 20UPGFC0016279 20UPGFC0016280 20UPGFC0016278 20UPGFC0016267 20UPGFC0016235 20UPGFC0016242 20UPGFC0016276 20UPGFC0016266 20UPGFC0016220 20UPGFC0016227
20UPGM20030001	20UPGB40500020	20UPGS83300003	20UPGPQ0030001	20UPGMC0210001	
20UPGM20030003	20UPGB40500021	20UPGS83300004	20UPGPQ0030003	20UPGMC0210002	
20UPGM20030006	20UPGB40500022	20UPGS83300001	20UPGPQ0030006	20UPGMC0210003	
20UPGM20030022	20UPGB40500023	20UPGS83300005	20UPGPQ0030022	20UPGMC0210004	
20UPGM20030024	20UPGB40500024	20UPGS83300006	20UPGPQ0030024	20UPGMC0210005	

図 7.4 登録した KEK モジュールのシリアルナンバー

ダウンロードしてくる必要がある。このような場合にはオプション 2 を使用すると考えられる。11 時間の処理時間要するような今のアルゴリズムでは運用は難しいと考えられるため、改善が必要である。

7.2.6 処理時間詳細

ダウンロード処理の詳細について以下の測定した。

1. 中央データベースからモジュール情報の取得
2. データベースでの FE チップ確認処理
3. 中央データベースからベアモジュール情報の取得
4. 中央データベースから FE チップ情報の取得 (4 枚分)
5. ローカルデータベースへの情報の書き込み (モジュール、FE チップ、品質試験情報)

情報取得のイメージを表??に示す。このように Quad モジュールの場合、ダウンロードの流れの中で合計して 7 回、データベース API を用いて情報取得を行う。

結果を表??に示す。

この結果より、各構成部品情報の取得 (モジュール、ベアモジュール、FE チップ) の取得にそれぞれ均等に処理時間がかかっていることがわかった。そのため、処理時間を改善するために、この情報取得の回数を減らすアルゴリズムを考える。

7.2.7 改善点の考案と見積もり

一括ダウンロード機能については以下の改善点が考えられる。

1. モジュールの現在位置に対応したものののみのダウンロード
2. FE チップの登録機関を取得しない
3. モジュールのプロパティとして、ダウンロードに必要な情報を全て保存

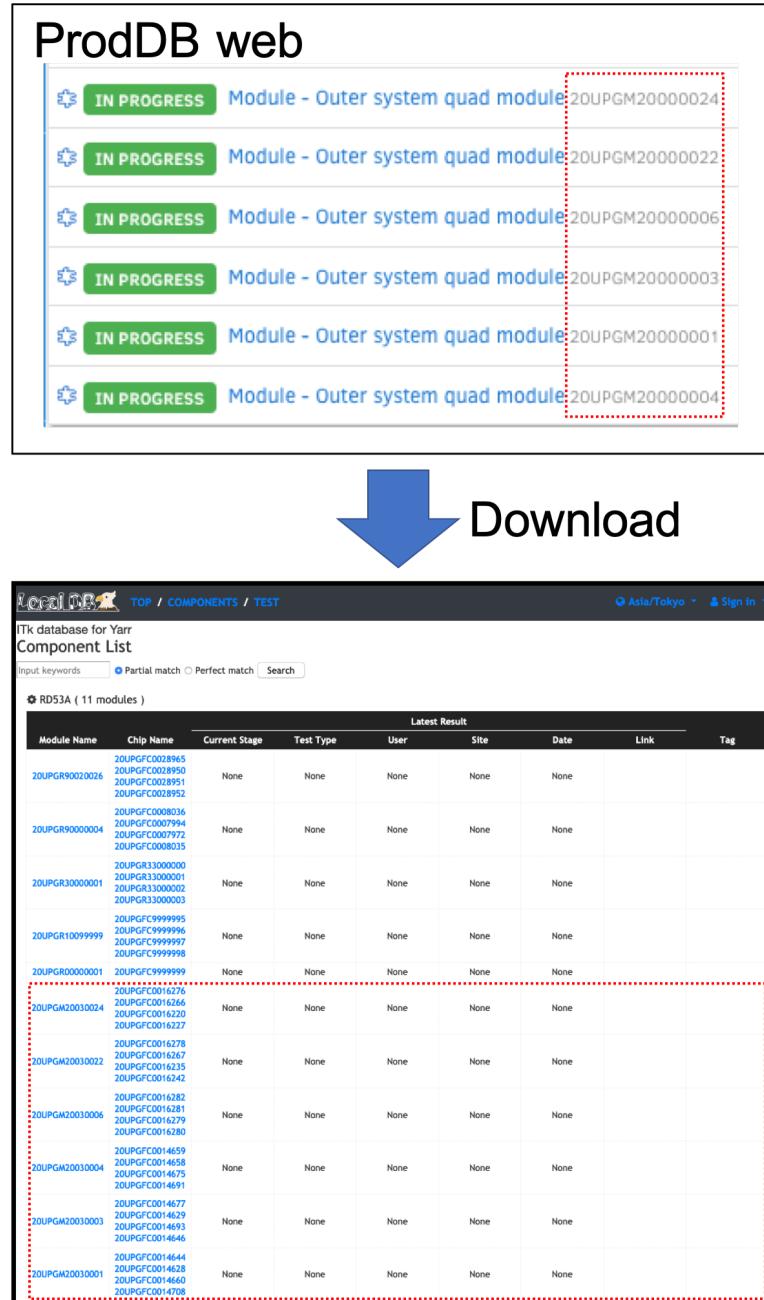


図 7.5 ダウンロードした KEK モジュールがアプリケーションで確認できている様子

4. データベース API を改良し、モジュール一覧取得の際に構成要素の情報を取得できるようにする

これらについて詳細と処理時間の見積もりを以下で行う。

改善案 1: モジュールの現在位置に対応したものののみのダウンロード

上述した機関が途中で変更となるような組み立ての流れにおいて、全てのモジュール ID をダウンロードする必要はない。中央データベースにはモジュールの現在位置情報を保持しているため、機能実行者と位置が同じもののみをダウンロードするアルゴリズムにすれば処理時間を改善できる。見積もりとしては、ダウンロード対象となるモジュール数を n とすると、以下のようになる。

$$(11 \pm 1) \times \frac{n}{10000} [\text{hour}] \quad (7.3)$$

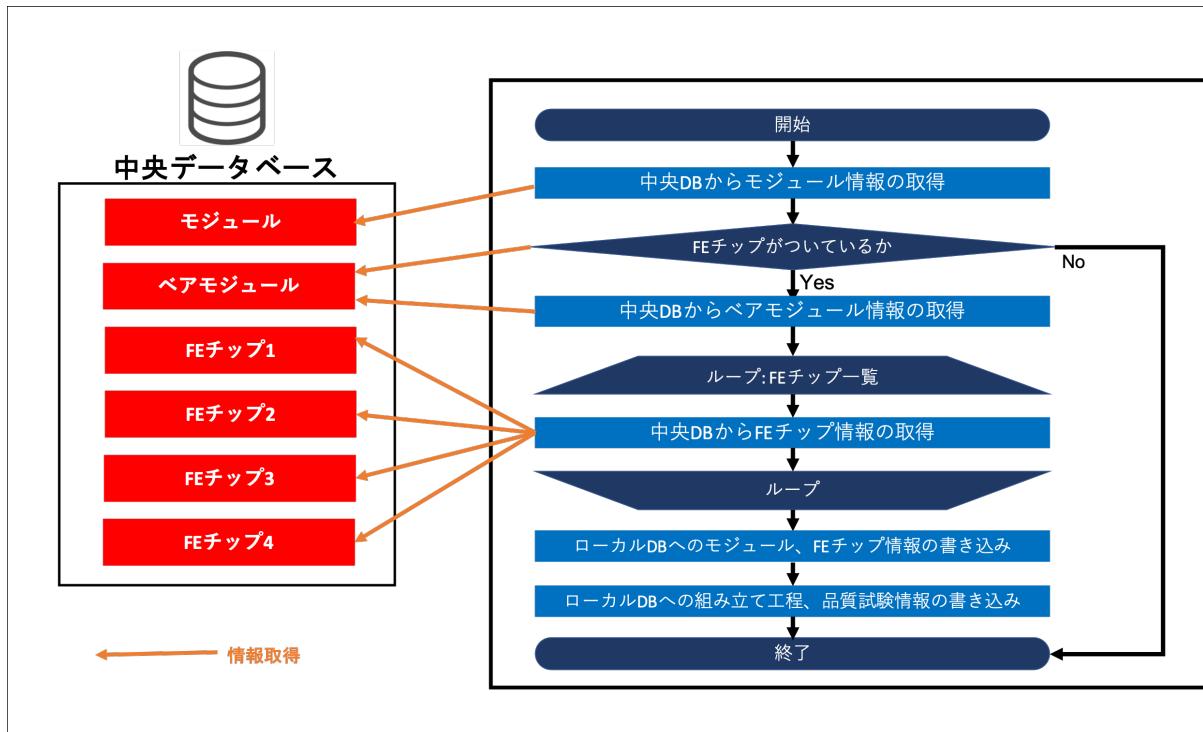


図 7.6 モジュール及び構成部品情報取得のイメージ

表 7.7 ダウンロード処理時間詳細

処理	時間
1	0.60 ± 0.07
2	0.55 ± 0.07
3	0.61 ± 0.04
4	0.46 ± 0.04
	0.71 ± 0.19
	0.51 ± 0.04
	0.57 ± 0.12
5	0.0025 ± 0.0011
合計	4.0 ± 0.1

改善案 2: FE チップの登録機関を取得しない

表??よりダウンロードの際に、FE チップの情報取得を行っている。これは FE チップ登録機関の情報を取得しローカルデータベースに保存するためであるが、登録機関の情報は組み立て現場で扱う作業としては、必要な情報ではない。そのため、現段階では FE チップのデータ取得処理は割愛することができる。これにかかる処理時間は表??より、合計して 2.3 ± 0.2 [sec] となるため、その場合オプション 2 の処理時間の見積もりは、以下のようになる。

$$\{(4.0 \pm 0.4) - (2.3 \pm 0.2)\} \times 10,000 = (1.8 \pm 0.3) \times 10^4 [\text{sec}] \quad (7.4)$$

$$= 4.9 \pm 0.8 [\text{hour}] \quad (7.5)$$

この改善策のデメリットとしては、FE チップの情報取得処理を省くとローカルデータベースで扱いたい情報が将来的にできた場合に保存できないことである。例えば各 FE チップの最適動作電圧のようにモジュール読み出しに対して有益な情報は保存し、迅速に確認したいという方針になる可能性もある。

改善案 3: モジュールのプロパティとして、ダウンロードに必要な情報を全て保存

モジュールのプロパティとして、FE チップの名前等のダウンロードに必要な情報を書いておくと、表??における listComponents によるモジュール一覧取得でその情報を参照することができる。こうすることで、表??において、ペアモジュールや FE チップの情報取得を省くことができる。この場合処理時間は、合計して $2.9 \pm 0.2[\text{sec}]$ となるため、その場合オプション 2 の処理時間の見積もりは、

$$\{(4.0 \pm 0.4) - (2.9 \pm 0.2)\} \times 10,000 = (1.1 \pm 0.3) \times 10^4[\text{sec}] \quad (7.6)$$

$$= 3.1 \pm 0.8[\text{hour}] \quad (7.7)$$

このデメリットは、データベースの中でデータが冗長になってしまうことである。FE チップの名前情報がモジュールのプロパティにも保存されていると、データベース内部で冗長性を持ってしまい編集が加えられた場合など、これを管理するのが難しくなる。

改善案 4:データベース API を改良し、モジュール一覧取得の際に構成要素の情報を取得できるようにする

現在、表??の listComponents を用いた時にはモジュール一覧の情報は取得できるが、各モジュールに対する構成要素は取得できない。そのため表??のようにモジュールごとに中央データベースに接続し、部品情報を取得している。ダウンロードに必要な情報を listComponents で一括で取得できるような仕様に変更を行えば、接続は一回ですみ、処理時間を削減できると考えられる。この場合、中央データベースの内部構造を知り、一括で取得しデータ送信をする場合にどれだけの時間を要するかを見積もり検討する必要がある。

現段階では組み立ての試験段階であり、現場で必要な情報、世界各地での組み立て工程の流れ等を検討している段階である。ここで述べたような改善策を組み合わせ、必要に応じて変更を加えていく必要がある。

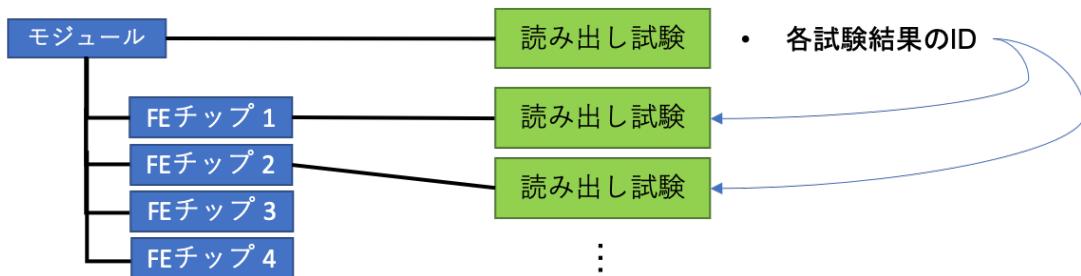


図 7.7 中央データベースにおける読み出し試験結果の構造

表 7.8 ダウンロード処理時間測定

部品	試験情報、結果	添付ファイル
モジュール	モジュール環境温度 FE チップにつく読み出し試験結果の ID	
FE チップ	ピクセル解析結果	試験結果データファイル 読み出し設定ファイル その他設定ファイル

7.3 読み出し試験結果のアップロード機能確認と処理時間測定

4章で述べたように、読み出し試験結果について中央データベースへアップロードするツールを開発した。以下で詳細を述べる。

7.3.1 アップロードする情報とその構造

読み出し試験結果について、中央データベースにアップロードする情報を以下に記す。

- 試験日時
- モジュール周りの環境温度
- ピクセル解析結果
- 各試験結果データファイル
- 読み出し設定ファイル
- その他設定ファイル (DB、ユーザ、組み立て機関等)

中央データベースにおける読み出し試験の構造に関して、YARR を用いて行った読み出し結果は全て FE チップ毎に取得、保存される。そのため、データベースの内部でも FE チップに読み出し試験結果を紐つける構造を設け、モジュールの結果では各 FE チップの結果ページの ID を持つ構造とした。イメージを図??に示す。

中央データベースにおいてモジュール、FE チップの試験結果が持つ情報を表??にまとめる。

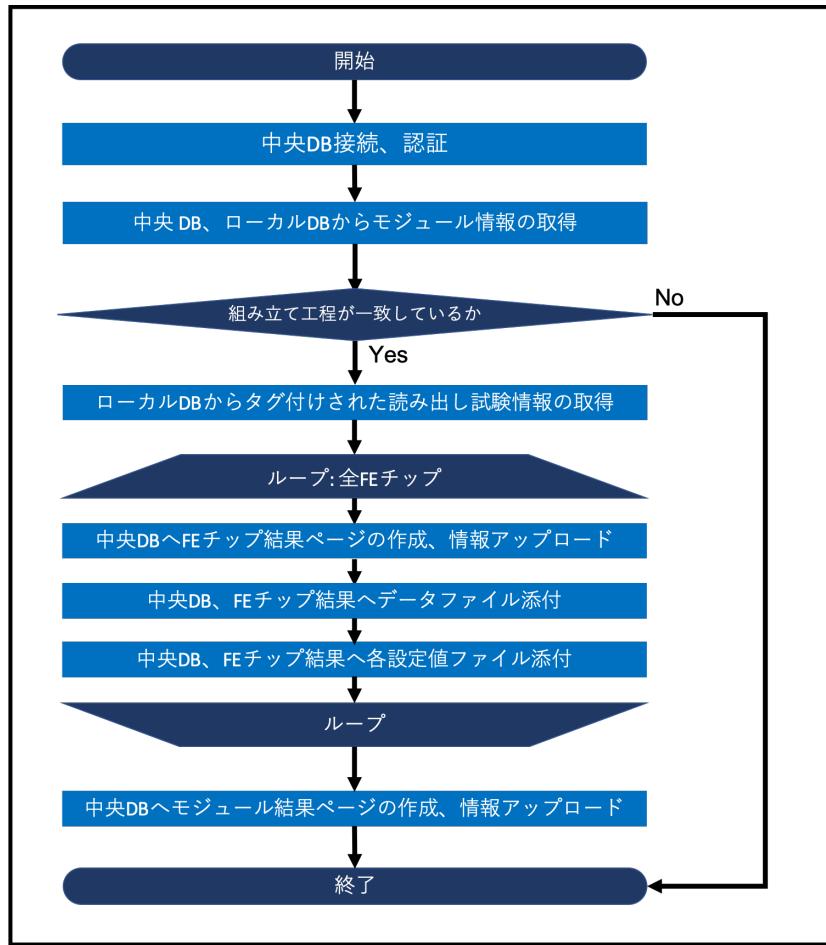


図 7.8 アップロード処理に関する流れのイメージ図

7.3.2 処理の流れ

アップロード機能における処理の流れのイメージを図??に示す。流れの中には共通して行われる処理と、各 FE チップに対して行われる処理がある。

7.3.3 機能確認と問題点

上述した処理のツールを開発し、機能確認と処理時間測定を行った。その際以下のようないくつかの問題があった。

- 処理時間が長くかかってしまった。
- ファイルの容量制限により、データ容量が 4MB を超えるファイルの添付に失敗した。

初めに、問題 1 の改善に向けて、アップロードにかかる時間を測定した。KEK のサーバーを用いてアップロード処理を 20 回行い、全体でかかる時間を測定した。平均値と標準偏差を測定値と誤差とした。以下のようにになった。

$$(1.3 \pm 0.0) \times 10^2 [\text{sec}] \quad (7.8)$$

ここで処理流れの表??より特に以下の詳細処理を抜粋し、それぞれにかかる時間を測定した。

表 7.9 アップロード処理詳細結果

処理	時間 [sec]
1	14 ± 0
2	0.88 ± 0.14
3	2.2 ± 0.1
4	68 ± 3
5	44 ± 0
6	1.8 ± 0.1

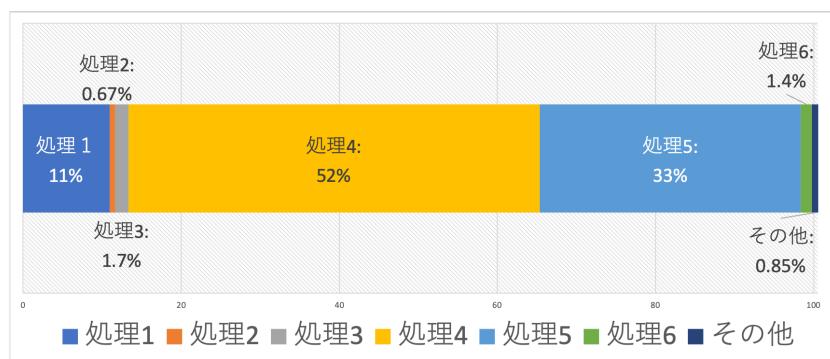


図 7.9 アップロード時における各詳細処理が占める割合

1. 中央データベース接続、認証。
2. 中央データベース、ローカルデータベースからモジュール情報の取得。
3. 中央データベースに FE チップ結果ページの作成。結果情報をアップロード。
4. 2 で作成した結果に対して、各データファイルを添付。
5. 2 で作成した結果に対して、各設定値ファイルを添付。
6. 中央データベースにモジュールの結果ページの作成。結果情報をアップロード。

結果を表??に示す。

それぞれの処理が全体に占める割合をグラフにしたものと表??に示す。結果データや各設定値のファイル添付に大きく時間がかかることがわかった。

生産時における見積もり

Quad モジュールにおける読み出し試験結果アップロード処理合計時間の見積もりを行った。上述した測定は SCC であるため FE チップに対する処理は 1 回であるため、Quad モジュールの場合は表??を用いて以下のように計算できる。

$$\text{FE チップ処理} : ((2.2 + 68 + 44) \pm \sqrt{(0.1)^2 + 3^2 + 0^2}) \times 4 \quad (7.9) \\ = (4.6 \pm 0.1) \times 10^2 [\text{sec}]$$

$$\text{合計} : (4.6 \pm 0.1) \times 10^2 + (14 \pm 0) + (0.88 \pm 0.14) + (1.8 \pm 0.1) = (4.7 \pm 0.1) \times 10^2 [\text{sec}] \quad (7.10) \\ = 7.9 \pm 0.1 [\text{min}]$$

モジュール読み出し試験 1 回に対して、約 8 分程度かかる見積もりとなった。円滑なモジュール組み立て、データ管理を行うために、データ同期は速やかに行われる必要がある。さらに同期が必要な品質試験

結果は、各組み立て工程に読み出し試験以外にも多く存在する。よって現状のアルゴリズムを見直し、処理速度を改善に努める必要があると考えた。

7.3.4 改善策

ファイル添付に多く時間がかかってしまっている現状を踏まえ、各ファイルに対して添付処理時間の測定を行った。測定は上述したものと同様に KEK サーバーを用いて合計 20 回行った。添付する結果データファイル、設定ファイルの種類とデータ容量、添付処理実行結果、処理時間を表??に示す。ここで、さらに 4MB を超える容量のファイル添付は失敗していることがわかった。

表 7.10: アップロード処理時のファイル添付実行結果

読み出し項目	ファイル名	実行結果	容量 [KB]	処理時間 [sec]	全体 [sec]
std_digitalscan	EnMask.json	Ok	1,300	3.3 ± 0.1	17±0
	OccupancyMap.json	Ok	1.500	2.9 ± 0.2	
	L1Dist.json	Ok	0.53	0.75 ± 0.12	
	ctrlCfg_ctrlCfg.json	Ok	0.46	0.61 ± 0.08	
	dbCfg_dbCfg.json	Ok	0.60	0.69 ± 0.16	
	siteCfg_siteCfg.json	Ok	0.033	0.61 ± 0.06	
	userCfg_userCfg.json	Ok	0.14	0.66 ± 0.09	
	scanCfg_std_digitalscan.json	Ok	2.2	0.55 ± 0.06	
	beforeCfg_chipCfg.json	Error	7,200	3.0 ± 0.2	
	afterCfg_chipCfg.json	Error	7,200	4.0 ± 0.2	
std_analogscan	EnMask.json	Ok	1,300	3.9 ± 0.1	17±0
	OccupancyMap.json	Ok	1.400	2.6 ± 0.1	
	L1Dist.json	Ok	0.60	0.69 ± 0.16	
	ctrlCfg_ctrlCfg.json	Ok	0.46	0.54 ± 0.05	
	dbCfg_dbCfg.json	Ok	0.60	0.49 ± 0.04	
	siteCfg_siteCfg.json	Ok	0.033	0.48 ± 0.04	
	userCfg_userCfg.json	Ok	0.14	0.58 ± 0.08	
	scanCfg_std_analogscan.json	Ok	2.1	0.45 ± 0.03	
	beforeCfg_chipCfg.json	Error	7,200	2.9 ± 0.2	
	afterCfg_chipCfg.json	Error	7,200	3.9 ± 0.3	
std_thresholdscan	Scurve-30-96.json	Ok	0.98	1.3 ± 0.1	49±1
	Scurve-110-96.json	Ok	0.98	0.45 ± 0.03	
	Scurve-70-96.json	Ok	0.98	0.47 ± 0.04	
	Scurve-150-96.json	Ok	1.0	0.46 ± 0.04	
	Scurve-190-96.json	Ok	1.0	0.64 ± 0.12	
	Scurve-230-96.json	Ok	1.0	0.49 ± 0.03	
	Scurve-270-96.json	Ok	1.0	0.47 ± 0.04	
	Scurve-310-96.json	Ok	1.0	0.47 ± 0.04	
	Scurve-350-96.json	Ok	1.0	0.49 ± 0.03	

表 7.10: アップロード処理時のファイル添付実行結果

Scurve-390-96.json	Ok	1.0	0.52 ± 0.06
Scurve-40-96.json	Ok	1.0	0.46 ± 0.03
Scurve-80-96.json	Ok	0.99	0.68 ± 0.13
Scurve-120-96.json	Ok	1.0	0.54 ± 0.07
Scurve-160-96.json	Ok	1.0	0.51 ± 0.05
Scurve-200-96.json	Ok	1.0	0.49 ± 0.04
Scurve-240-96.json	Ok	1.0	0.50 ± 0.05
Scurve-280-96.json	Ok	1.0	0.48 ± 0.04
Scurve-320-96.json	Ok	1.0	0.49 ± 0.05
Scurve-360-96.json	Ok	1.0	0.49 ± 0.06
Scurve-400-96.json	Ok	1.0	0.45 ± 0.05
Scurve-10-96.json	Ok	1.0	0.42 ± 0.03
Scurve-50-96.json	Ok	0.99	0.49 ± 0.05
Scurve-90-96.json	Ok	0.99	0.46 ± 0.05
Scurve-130-96.json	Ok	1.0	0.47 ± 0.05
Scurve-170-96.json	Ok	1.0	0.52 ± 0.04
Scurve-210-96.json	Ok	1.0	0.51 ± 0.04
Scurve-250-96.json	Ok	1.0	0.58 ± 0.10
Scurve-290-96.json	Ok	1.0	0.64 ± 0.13
Scurve-330-96.json	Ok	1.0	0.64 ± 0.09
Scurve-370-96.json	Ok	1.0	0.49 ± 0.06
Scurve-60-96.json	Ok	0.99	0.51 ± 0.06
Scurve-100-96.json	Ok	1.0	0.48 ± 0.05
Scurve-140-96.json	Ok	1.0	0.48 ± 0.06
Scurve-180-96.json	Ok	1.0	0.52 ± 0.06
Scurve-220-96.json	Ok	1.0	0.54 ± 0.05
Scurve-260-96.json	Ok	1.0	0.51 ± 0.05
Scurve-300-96.json	Ok	1.0	0.66 ± 0.09
Scurve-340-96.json	Ok	1.0	0.51 ± 0.06
Scurve-380-96.json	Ok	1.0	0.55 ± 0.05
sCurve-0.json	Ok	49	1.0 ± 0.1
ThresholdDist-0.json	Ok	4.6	0.56 ± 0.06
ThresholdMap-0.json	Ok	2,200	4.3 ± 0.1
NoiseDist-0.json	Ok	2.3	0.42 ± 0.04
Chi2Map-0.json	Ok	2,300	4.5 ± 0.1
StatusMap-0.json	Ok	1,300	2.8 ± 0.1
StatusDist-0.json	Ok	0.49	0.48 ± 0.04
NoiseMap-0.json	Ok	2,200	4.0 ± 0.2
Chi2Dist-0.json	Ok	1.1	0.50 ± 0.04

表 7.10: アップロード処理時のファイル添付実行結果

	TimePerFitDist-0.json	Ok	3.1	0.56 ± 0.12	
	ctrlCfg_ctrlCfg.json	Ok	0.46	0.49 ± 0.05	
	dbCfg_dbCfg.json	Ok	0.60	0.52 ± 0.06	
	siteCfg_siteCfg.json	Ok	0.033	0.54 ± 0.07	
	userCfg_userCfg.json	Ok	0.14	0.60 ± 0.07	
	scanCfg_std_thresholdscan.json	Ok	2.2	0.46 ± 0.03	
	beforeCfg_chipCfg.json	Error	7,200	3.2 ± 0.2	
	afterCfg_chipCfg.json	Error	7,200	3.5 ± 0.1	
std_totscan	MeanTotMap-0.json	Ok	1,900	4.8 ± 0.1	20±0
	SigmaTotMap-0.json	Ok	2,200	4.1 ± 0.2	
	MeanTotDist-0.json	Ok	0.59	0.54 ± 0.07	
	SigmaTotDist-0.json	Ok	1.8	0.47 ± 0.03	
	L1Dist.json	Ok	0.59	0.60 ± 0.08	
	ctrlCfg_ctrlCfg.json	Ok	0.46	0.47 ± 0.03	
	dbCfg_dbCfg.json	Ok	0.60	0.67 ± 0.24	
	siteCfg_siteCfg.json	Ok	0.033	0.59 ± 0.10	
	userCfg_userCfg.json	Ok	0.14	0.56 ± 0.05	
	scanCfg_std_totscan.json	Ok	2.0	0.59 ± 0.10	
	beforeCfg_chipCfg.json	Error	7,200	3.0 ± 0.1	
	afterCfg_chipCfg.json	Error	7,200	3.9 ± 0.3	
std_noisescan	Occupancy.json	Ok	1,300	4.0 ± 0.1	18±0
	NoiseOccupancy.json	Ok	1,300	2.5 ± 0.1	
	NoiseMask.json	Ok	1,300	2.4 ± 0.1	
	ctrlCfg_ctrlCfg.json	Ok	0.46	0.57 ± 0.08	
	dbCfg_dbCfg.json	Ok	0.60	0.55 ± 0.06	
	siteCfg_siteCfg.json	Ok	0.033	0.53 ± 0.04	
	userCfg_userCfg.json	Ok	0.14	0.61 ± 0.07	
	scanCfg_std_noisescan.json	Ok	1.4	0.53 ± 0.05	
	beforeCfg_chipCfg.json	Error	7,200	2.8 ± 0.2	
	afterCfg_chipCfg.json	Error	7,200	3.5 ± 0.1	

表??より、データサイズの大きいものにアップロード時間がかかっていることがわかる。また添付処理を行うオフセットがあることから、threshold scan のように各容量が大きくなくてもファイル数が多いものにはアップロード時間が合計して多くかかってしまうことがわかる。

これらのことと添付処理の失敗をなくすことを考慮に入れ、次のような改善策を考えた。

- 各試験項目に対する結果データ、設定ファイルをそれぞれ Zip ファイルに統合し、圧縮後にアップロードを行う。

こうすることで、アップロードするファイルの容量、数共に削減することができる。圧縮率によってはアップロード処理の失敗もなくすことができると考えた。

これを踏まえアップロードツールを改良し、再び各ファイルの添付処理にかかる時間を測定した。合計処理時間は以下のようになり、全てのファイルのアップロードに成功した。

$$36 \pm 1[\text{sec}] \quad (7.11)$$

表 7.11: アップロード処理時の zip ファイル添付実行結果

読み出し項目	ファイル名	実行結果	容量 [KB]	処理時間 [sec]	全体 [sec]
std_digitalscan	std_digitalscan_datafiles.zip	Ok	10	0.77 ± 0.18	1.9 ± 0.2
	std_digitalscan_configfiles.zip	Ok	56	1.1 ± 0.1	
std_analogscan	std_analogscan_datafiles.zip	Ok	46	1.0 ± 0.2	2.2 ± 0.3
	std_analogscan_configfiles.zip	Ok	58	1.2 ± 0.2	
std_thresholdscan	std_thresholdscan_datafiles.zip	Ok	1,500	2.6 ± 0.1	3.5 ± 0.1
	std_thresholdscan_configfiles.zip	Ok	190	0.86 ± 0.08	
std_totscan	std_totscan_datafiles.zip	Ok	730	1.7 ± 0.2	2.5 ± 0.2
	std_totscan_configfiles.zip	Ok	190	0.83 ± 0.15	
std_noisescan	std_noisescan_datafiles.zip	Ok	19	0.56 ± 0.07	1.7 ± 0.1
	std_noisescan_configfiles.zip	Ok	190	1.1 ± 0.1	

生産時における見積もり

上記の見積もりと同様に、改善後のツールにおけるアップロード時間の見積もりを行った。表??において、添付処理に対応する処理 4、5 以外は同じとする。改良後の処理 4、5 の処理時間は、

$$\begin{aligned} \text{FE チップ処理} : ((2.2 + 6.7 + 5.1) &\pm \sqrt{(0.1)^2 + (1.1)^2 + (0.3)^2}) \times 4 \\ &= 56 \pm 5[\text{sec}] \end{aligned} \quad (7.12)$$

$$\begin{aligned} \text{合計} : (56 \pm 5) + (14 \pm 0) + (0.88 \pm 0.14) + (1.8 \pm 0.1) &= 73 \pm 5[\text{sec}] \\ &= 1.2 \pm 0.1[\text{min}] \end{aligned} \quad (7.13)$$

約 1 分でアップロードを完了できる見積もりとなった。改善前に比べて 15% の処理時間となった。現在は改善後の方針を用いたツールを提供している。

第8章

まとめ

8.1 まとめ

8.2 SW リリース

8.3 今後の課題

8.4 結論

付録 A

ローカルデータベースにおける読み出し 試験結果検索システムの性能評価

付録 B

読み出し試験に用いたハードウェア詳細

付録がいる場合はどうぞ。

付録 C

ファイル送信時におけるデータサイズと処理時間の関係について

まず線形にはならない。高速化するために、いろいろ工夫してある。

Server の spec は同程度。Cubic、Window size と ping(111msec) は変わらない。一般的には上りより下りの方が太い。よって、KEK の上り network は LBL 上りと比べての方が細いことがわかる。

KEK の上りが LBL に比べて細い。それに加えて RTT が KEK の方が大きいため遅延が広がる。

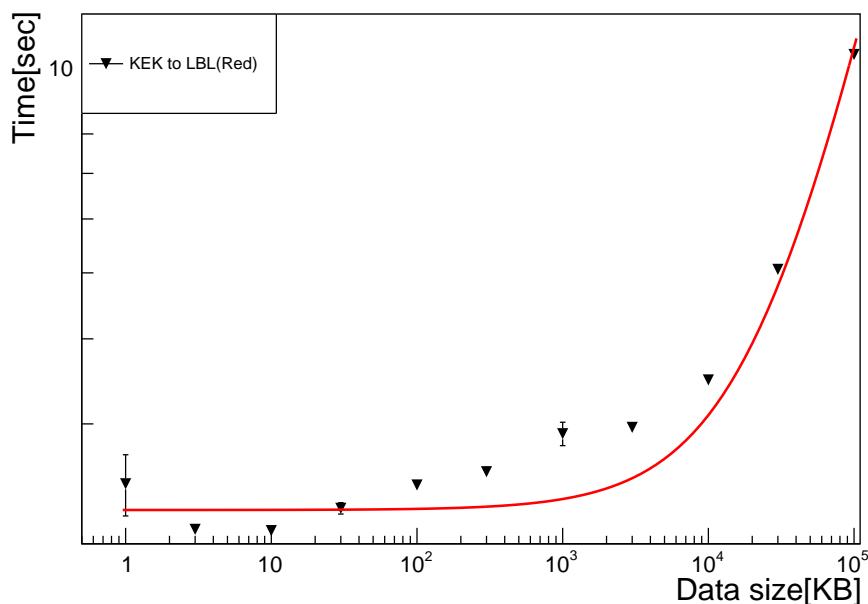


図 C.1 添付するファイルサイズと処理時間の関係

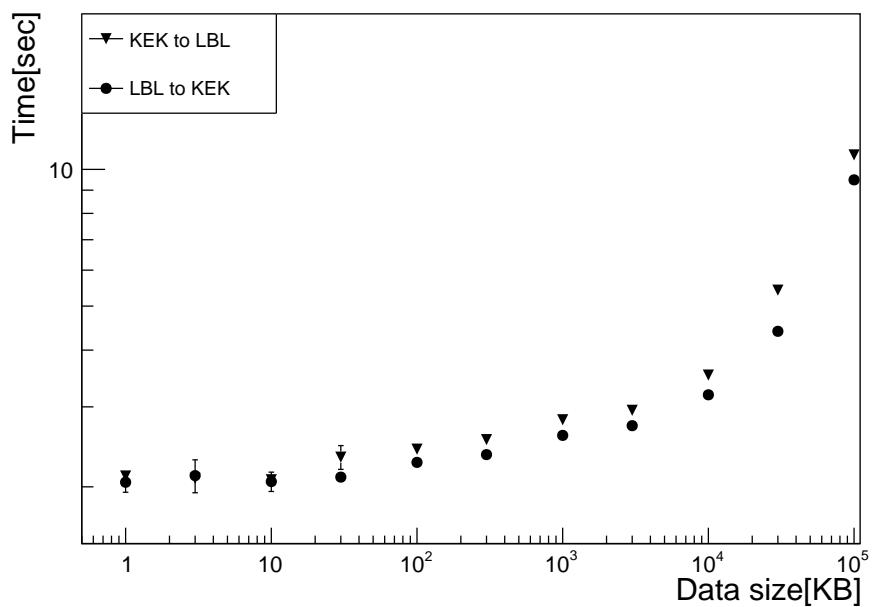


図 C.2 KEK、LBL 間のファイル送信

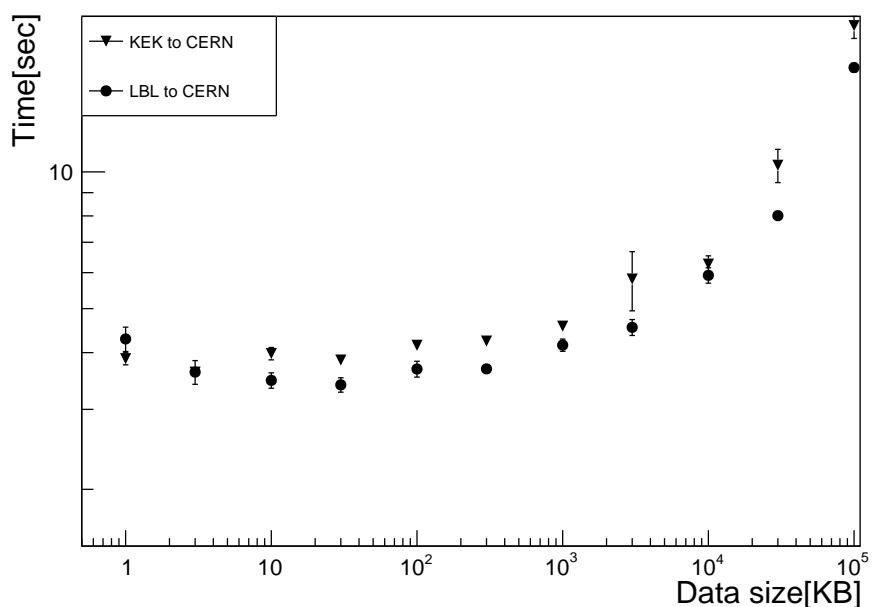


図 C.3 KEK、LBL と CERN 間のファイル送信

参考文献

[テキスト] 東京工業大学理学部物理学科『物理実験学第一』(2009)

謝辞